UNIVERSITAT DE BARCELONA

FUNDAMENTAL PRINCIPLES OF DATA SCIENCE MASTER'S THESIS

Deep learning for predictive maintenance of rolling bearings

Author:
Laia Domingo Colomer

Supervisor:
Dr. Jordi Vitrià

A thesis submitted in partial fulfillment of the requirements for the degree of MSc in Fundamental Principles of Data Science

in the

Facultat de Matemàtiques i Informàtica

June 3, 2020

UNIVERSITAT DE BARCELONA

Abstract

Facultat de Matemàtiques i Informàtica

MSc

Deep learning for predictive maintenance of rolling bearings

by Laia Domingo Colomer

The monitoring of machine health has become of great importance in the industry in the recent years. Unexpected equipment failures can lead to catastrophic consequences, such as production downtime and costly equipment replacement. Rolling bearings are one of the most delicate components of rotating equipment, being a common cause of machine failures. For this reason, predictive maintenance techniques of rolling bearings are fundamental to preserve the health of a machine. In this project, we present a deep learning approach to predict bearing failures in their early development. All methodologies are data-driven, therefore they do not assume any expert knowledge on the field nor require any information about the equipment's operating conditions. For this reason, this approach is versatile and can be used to diagnose multiple machines.

Acknowledgements

I would like to express my deep gratitude to my supervisor Dr. Jordi Vitrià, for his valuable support, suggestions and ideas during the development of this project. His enthusiastic dedication to the project allowed me to enjoy and learn from this experience.

I would also like to thank my Accenture supervisor Dr. Jesús Gabaldón for his unconditional support and for helping me understand the business perspective. This project would not have been possible without his help and efforts.

I would also like to thank my Accenture colleagues Paula Lopes Romero and Jan Hayer for making me feel part of their community and sharing their expertise with me.

Finally I wish to thank my parents, partner and friends for their unconditional support and encouragement throughout this study.

Contents

Introduction

Predictive maintenance is a set of techniques which use condition-monitoring tools to control the performance of a machine and detect possible defects before they fail. This set of methods are crucial to apply appropriate maintenance to equipment. Too early maintenance can cause spending money unnecessarily, while too late maintenance can cause catastrophic damage to equipment.

Rolling bearings are critical components used extensively in rotating equipment and, if they fail unexpectedly, can result in a catastrophic failure with associated high repair and replacement costs. Therefore, monitoring their health status is of great interest. Moreover, the growing availability and lowering costs of sensor devices and associated technology have enabled companies to install sensor devices on all key equipment and machinery. This motivates the use of data-driven approaches to predict bearing failures in rotating equipment.

The goal of this project is to provide a solution to predict the bearing health status. More precisely, we aim to design a deep learning model which can detect the early development of a bearing failure. Moreover, the model should be able to tell the component of the bearing where the failure is happening. This approach is data-based, which means that it assumes nothing about the physical model of the system, in contrast to model-based approaches, which require expert knowledge on the system behaviour. The code from all the experiments can be found in https://github.com/laiadc/PFM_Bearing_Fault_Detection.

The report is organised as follows. Chapter 1 offers an introduction to rolling element bearings and describes the problem we aim to solve. Chapter 2 provides a mathematical description of all the models and techniques used in this project. Chapter 3 describes the data sets used to train the models and provides implementation details for such models. We proceed in Chapter 4 with the implementation of the algorithms and the assessment of the results. Finally, we conclude in Chapter 5 with the conclusions of this project.

Chapter 1

Problem introduction

The goal of this chapter is to present the problem that will be treated in this project. In particular, this chapter begins with an introduction to rolling element bearings and possible failures. Then, some fault diagnosis techniques are discussed. Finally, the challenges and requirements of the problem are explained.

1.1 Introduction to rolling element bearings

Rolling element bearings are mechanical elements present in most rotating machines. Their purpose is to reduce the rotational friction and to support axial and radial loads. Rolling bearings are critical components used extensively in rotating equipment and, if they fail unexpectedly, can result in a catastrophic failure with associated high repair and replacement costs. In fact, about 40% of inductive industrial motor failures result from bearing failures [1].

Rolling elements consist of four different components: outer ring, rolling element, cage and inner ring, as shown in figure 1.1.

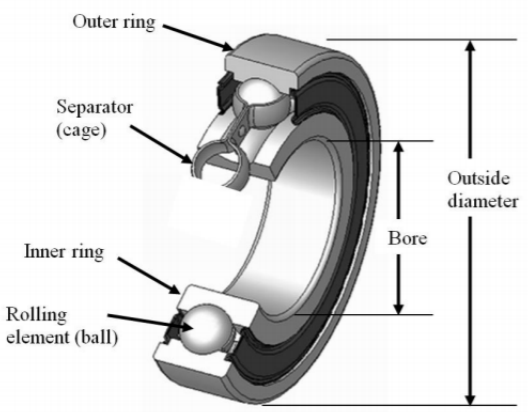


FIGURE 1.1: Rolling element bearing components. Figure from [1]

In general, rolling element bearings have low starting friction and remain unaffected by operating temperature. Therefore, if the installation and maintenance is performed correctly, the usual cause of bearing failures is metal fatigue. However, as bearings usually undergo harsh conditions, other sources of failure exist. Some examples of bearing failures sources are loss of lubrication, corrosion, misalignment, overheating, excessive loading and contamination [2].

1.1.1 Failure identification

Vibration analysis is decisive in diagnosing the damage of a rolling element bearing. By comparing the evolution of the spectral amplitudes, one can predict the future degree of deterioration. A failure in one of the bearing components will produce a pronounced peak in a characteristic frequency of the frequency spectrum. This characteristic frequency will allow quick and easy identification of the failure. The four possible bearing failing frequencies (or characteristic frequencies) are [3]:

- **Ball Pass Frequency Outer (BPFO):**
  Corresponds to the number of balls that pass through a given point of the outer ring during a full rotation shaft.

$$\text{BPFO} = \frac{f_r \cdot N_B}{2}\left(1 - \frac{d}{D}\cos\alpha\right) \tag{1.1}$$

- **Ball Pass Frequency Inner (BPFI):**
  Corresponds to the number of balls that pass through a given point of the inner ring during a full rotation shaft.

$$\text{BPFI} = \frac{f_r \cdot N_B}{2}\left(1 + \frac{d}{D}\cos\alpha\right) \tag{1.2}$$

- **Ball Spin Frequency (BSF):**
  Number of turns a roller makes during a full rotation shaft.

$$\text{BSF} = \frac{f_r \cdot D}{2d}\left(1 - (\frac{d}{D}\cos\alpha)^2\right) \tag{1.3}$$

- **Fundamental Train Frequency (FTF):**
  Number of turns a bearing cage makes during a full rotation shaft.

$$\text{FTF} = \frac{f_r}{2}\left(1 - \frac{d}{D}\cos\alpha\right) \tag{1.4}$$

where f_r is the shaft speed, N_B is the number of rolling elements (or balls), α is the angle of the load from the radial plane, and $D = \frac{D_1+D_2}{2}$. These quantities are illustrated in Figure 1.2. Usually, instead of having the bearing failing frequencies, the bearing failing coefficients are given. These coefficients are defined as the bearing failing frequencies divided by the shaft speed f_r.

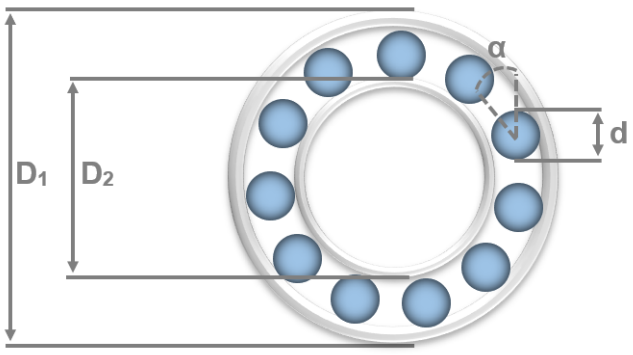


FIGURE 1.2: Rolling element bearing failing frequencies.

As mentioned above, the bearing failing frequencies can be used to diagnose bearing failures. Figure 1.3 illustrates how bearing failure modes can be detected using vibration analysis. A failure mode in a certain bearing component will appear as a pronounced peak in its characteristic frequency. This peak will be seen in the frequency spectrum of the signal. Two amplitude thresholds will be defined for each bearing component. These thresholds are specific of each machine and can even be discussed with the client. Then, if the amplitude of a certain peak is above the first threshold, a warning alarm is sent. If the amplitude peak is above the second threshold, a critical alarm is sent.

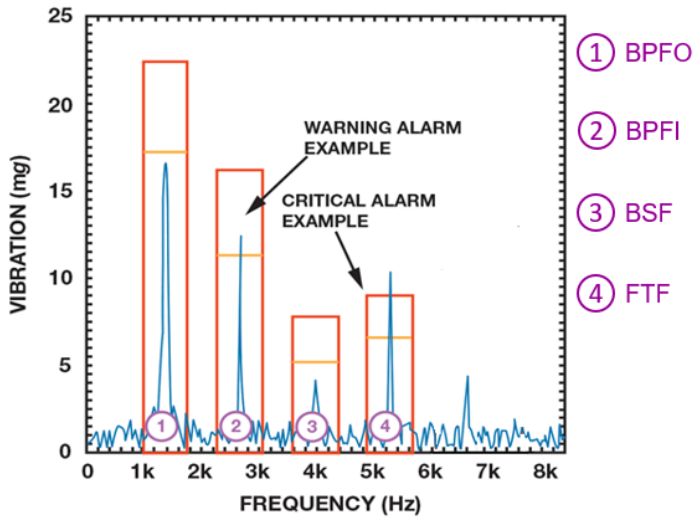


FIGURE 1.3: Example of failure detection using vibration analysis.

1.1.2 Bearing damage severity level

If the bearings have been properly installed, outer ring failures are the most frequent failures, followed by inner ring, rolling elements and finally cage failures. Bearings with defects in the outer ring generally have a longer expected life than bearings with defects in the inner ring. Moreover, the amplitude of the outer ring fault frequencies is usually higher than the amplitude of the inner ring fault. The explanation for this is that the sensors measuring the signal are mounted closer to the outer ring, hence a signal from the inner ring has first to travel through the rolling elements, the cage and the outer ring. This journey attenuates the signal, resulting in lower amplitude peaks. Bearing failures can usually be classified into three different deterioration stages [2]:

- In the initial phase, the deterioration is visible in high-frequency bands. The amplitude peaks tend to be low.
- In the intermediate phase, the harmonics of the high-frequency bands slowly increase its magnitude.
- In the final stage, background noise becomes visible in acceleration at high frequency. Moreover, new bearing failures may develop, indicating greater defect severity.

1.2 Fault diagnosis techniques

This section briefly describes multiple techniques that have been used to identify and classify bearing faults. Fault diagnosis approaches can be classified into two categories: model-based and data-based [4]. Model-based approaches try to mathematically model the underlying physical process in order to understand and predict the true behaviour of the system. An example of model-based technique is vibration analysis using the bearing failure frequencies, as described in section 1.1.1. These techniques require expert knowledge as well as some detailed information of the system. In the example of section 1.1.1, one needs to know both the shaft speed and the bearing failure coefficients to perform the spectrum analysis. On the other hand, data-based models do not require expert knowledge in the field and do not rely on understanding the physical process of the system. Such methodologies are receiving increased attention in recent years due to the abundance of data and the advancement of processing techniques.

Data based methodologies have been broadly studied over the years. Classical approaches in the time domain are time series models such as ARMA models, spectral Kurtosis and Singular-Value decomposition [5]. Most of these techniques are application-specific, which may be an inconvenient due to varying operational conditions.

To achieve a better performance at versatile operating conditions and noisy environments, machine learning and deep learning-based methods are becoming increasingly popular to meet this demand. In the field of machine learning, dimensionality reduction algorithms such as principal component analysis (PCA) [6] or singular value decomposition (SVD) are employed as feature extractors. Then, algorithms such as Random Forests [7], KNearest Neighbours [8] and Support Vector Machines [9] are employed to classify failure types. In the field of deep learning, Convolutional Neural Networks [10, 11] rose to popularity due to the power of the feature extraction layers. Also, Recurrent Neural Networks and Long Short-Term Memory Neural Networks [12, 13, 14] were used to predict the remaining useful time of bearings. This section has only provided a brief overview of the most popular methods in the fault diagnosis field. For a more detailed discussion see, [4] which provides an excellent overview of the current literature on studies that apply machine learning and deep learning techniques to bearing fault diagnosis.

1.3 Problem description

The goal of this project is to design a model to classify the health state of a rolling element bearing. That is, to identify whether there is a failure and if so, identify the position of such failure. Moreover, there are some further requirements, listed below:

- In the case where the shaft velocity is known and the bearing failure coefficients are provided, a model-based approach is enough to classify the bearing failure mode. Therefore, the goal of this approach is to be able to classify the bearing health state when the shaft velocity is unknown or even variable. The model must thus be **robust to different rotating velocities**.

- The **bearing failure coefficients** are also supposed to be **unknown**. Therefore, the approach must not use such coefficients as features to feed the model.
- The model must be able to **detect failure modes at early stage** so that one can take appropriate actions before the machine breaks.
- Usually, little historical data is available to train the models. Therefore, the model must be able to work with **small training data sets**.

1.3.1 Data structure

In the final section of this chapter, the structure of the data used for training and testing the model is presented.

In this project, we will use public data sets, from various universities or research centres. As we said in the previous section, we are interested in detecting the health state at the early stages of the failure development. For this reason, the data sets will contain vibration signal snapshots (for example 1-second recordings) at specific intervals of time. Signals were recorded using an accelerometer installed on the bearing housing. Hence, the vibration signal will be given as a waveform. That is, each snapshot contains the acceleration magnitude as a function of time. Figure 1.4 shows an example of a waveform.

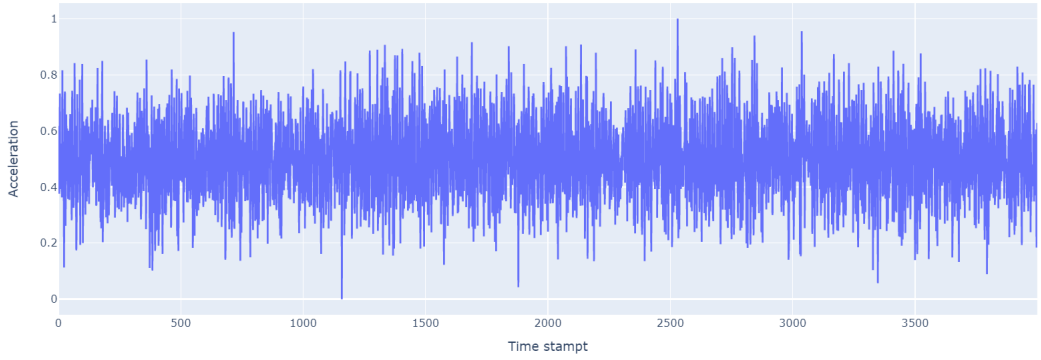


FIGURE 1.4: Example of a waveform snapshot.

Signals are recorded until the bearing failure is completely developed and the machine has to be stopped. Therefore, the only known information is the bearing failure at the end of the experiment, but we do not know when the failure started. Hence, we will have to look for some strategy that allows to label these data sets with the failure mode at the beginning of its development.

Finally, in order to ensure that the model is robust to different operation conditions (in particular to different rotating velocities), several data sets will be mixed during training.

Chapter 2

Mathematical description of the models

This chapter aims to provide some mathematical background on the models used in the project. According to the problem requirements described in section 1.3, the problem will be split into three subproblems:

1. **Early detection of the failure**
   Recall that one of the requirements is to be able to classify the bearing failure at the early stages of its development. For this reason, an early detection model will be developed. This model will be then used to put the labels to the training data for the bearing failures classification model. This model will be based on a one-dimensional convolutional autoencoder.

2. **Classification of the failure**
   Once the labels of the failure modes have been obtained, a classification model will be trained to determine the position (inner ring, outer ring, rolling element or cage) of the bearing failure. This model will be trained so that it is robust to different operation conditions (such as the rotation velocity) and so that it can be trained with few data samples. In order to accomplish these requirements, a convolutional neural network will be trained using triplet learning strategies or using a Siamese network with contrastive loss.

3. **Continuous learning of the models**
   Usually, all the historical data is not provided at the training stage, but it is acquired progressively with time. Therefore, it is interesting to be able to continuously train the model as new data arrives. This new data can contain unseen behaviour for the model. It is desirable to make the model learn this new behaviour without forgetting about the previously learnt experiences. To deal with this problem, we will use a technique called Elastic Weight Consolidation.

The following sections explain the mathematical concepts of these three subproblems and an introduction to neural networks.

2.1 First concepts

This section introduces the basic concepts that will be used in all the models. In particular, a brief introduction to artificial neural networks is provided. This section does not aim to provide deep explanations of these concepts, but to give context to the models used in the project. For further details, references are provided.

2.1.1 Artificial neural networks

Artificial Neural Networks (ANN) [15] are a widely used method for generalization problems, specifically for non-linear function approximation [16]. An ANN is an information processing paradigm that is inspired by the way biological nervous systems process information. It is composed of a large number of interconnected neurons, which work in unison to solve specific problems. ANN are considered to be *deep* if there are formed by a large number of neuron layers. In order to use an ANN for function approximation, it has to go through a training process, where examples are fed to the architecture.

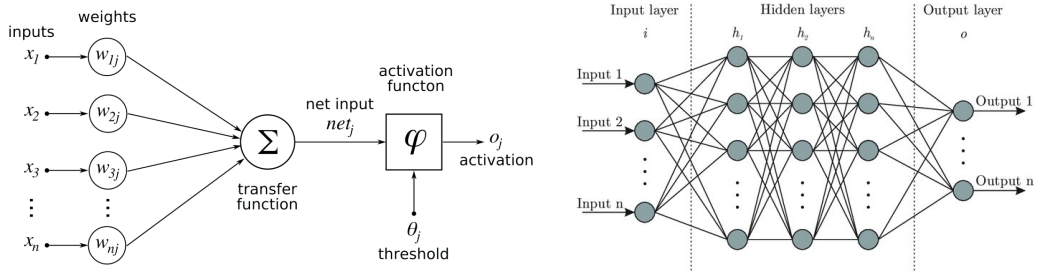


FIGURE 2.1: Representation of the architecture of a layer of a neural network (A) and a complete neural network (B). Figures from [17, 18]

A neuron can be described as a computational unit receiving input $x = (x_0, x_1, x_2, \cdots, x_m)$ (x_0 called the bias usually set to be +1) and outputs $h(x) = \varphi(Wx)$ with φ being an activation function and W being a matrix of the weights of the network. Usually, neurons are organised in layers with connections between two adjacent layers. The most common layer type is called the fully-connected layer (shown in Figure 2.1 (B)) in which all neurons in two adjacent layers are fully pairwise connected but neurons within the same layer share no connections.

The number of layers and neurons in each layer are hyper-parameters one has to tune. If the neural network is too shallow the network may not be capable of adequately learning the problem whereas too deep networks may not be able to generalise to new data due to having too high complexity. The neural network described in Figure 2.1 (A) can be written mathematically in the following way:

$$\begin{aligned} \Sigma_j &= W_j x + b \\ o_j &= h(x) = \varphi(\Sigma_j) \end{aligned}$$

where φ is the activation function, W_j is the matrix of weights for layer j and x is the input vector. o_j is an output neuron that, together with other output neurons $\{o_i\}$, will be part of the first hidden layer of the network. This hidden layer will take as an input the outputs $\{o_i\}$. This process is illustrated in Figure 2.1 (B).

2.1.2 Optimization

Finding the optimal parameters of a neural network can be formulated as an optimization problem, which tries to minimize a loss function. A loss function $\mathcal{L}(y, f(\mathbf{x})) = \frac{1}{n}\sum_i \ell(y_i, f(\mathbf{x_i}))$ represents the price paid for the inaccuracy of predictions in a classification or regression problem. Common loss functions are mean square error

(MSE) or mean absolute error (MAE) for regression tasks, and logistic loss and cross-entropy loss for classification tasks.

In order to minimize the loss function $\mathcal{L}$, many approaches have been proposed. The most popular method is called gradient descent, which updates the weight parameters in the direction of the negative gradient of the loss function. This direction is the one with the highest decrease in the loss function. For more information see [19].

Finally, the algorithm for calculating the gradients of the network is called backpropagation. The backpropagation [20] starts from the output layer calculating the gradients of the previous layer until the input layer. In general, the gradients are calculated based on the multivariate chain rule throughout the computational graph. The computation of derivatives is done by using *Automatic differentiation* (AD) [21, 22], which systematically applied the chain rule at the elementary operator level. AD relies on the fact that all numerical computations are ultimately compositions of a finite set of elementary operations for which derivatives are known. For this reason, given a library of derivatives of all elementary functions in a deep neural network, it is possible to compute the derivatives of the network with respect to all parameters at machine precision and applying stochastic gradient methods to its training. Without AD, the design and debugging of optimization processes for complex neural networks with millions of parameters would be impossible.

2.1.3 Convolutional neural networks

A type of deep ANN which has proven to produce very successful results in deep learning is the deep convolutional network. This type of network is specialised in processing high-dimensional data in the form of spatial arrays, such as time series (in 1D) or images (in 2D). The name stems from the fact that instead of general matrix multiplication it employs a mathematical convolution in at least one of its layers. Mathematically, given a convolution kernel K, or **filter**, represented by a $(M \times N)$ array, the convolution of an array A with K is:

$$output(x,y) = (A \otimes K)(x,y) = \sum_{m=0}^{M-1} \sum_{n=1}^{N-1} K(m,n)A(x-n, y-m)$$

The output of the convolution is another array that might represent some kind of information that was present in the initial array in a very subtle way. In this way, a filter of a convolutional neural network is responsible for detecting one feature of the network input. The kernel matrix are free parameters that must be learned to perform the optimal feature extraction. An example of convolution is shown in Figure 2.2 (A).

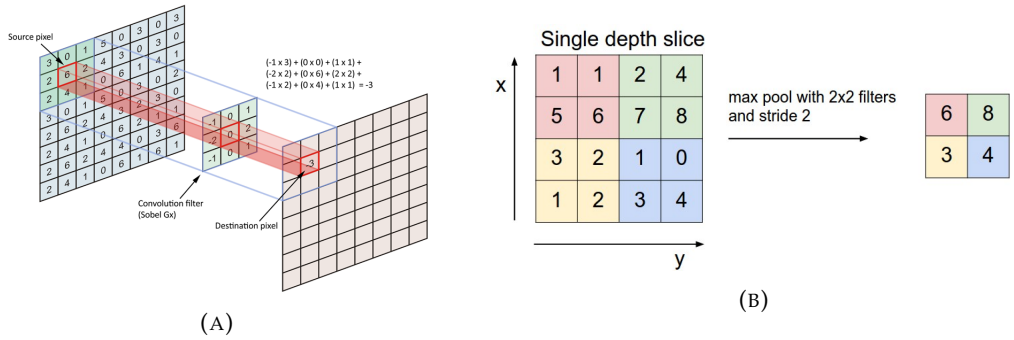


FIGURE 2.2: Convolutional filter (A) and a max pooling function (B). Figures from [23].

Another hyperparameter of the network is the stride size with which the kernel slides over the input. Typically the convolutional operation is followed by a non-linear activation function which adds non-linearity to the system. In the last step, a pooling layer is added in order to progressively reduce the spatial size of the array. The usual pooling functions are max pooling and average pooling, which are applied to the output of the filter convolution. An example of a max-pooling layer is shown in Figure 2.2 (B).

2.2 Early detection of the failure

As we have mentioned before, one of the important goals of the model is to be able to identify the failure mode at the early stages of its development. Therefore, to train a classification model, we need to have a labelled data set containing the full development of the failure. Since our data only contains the bearing failure which has appeared at the end of the experiment, we need to find a way to estimate the beginning of such failure. In order to do so, we will use an unsupervised method which detects if there has been a deviation from the normal (healthy) behaviour of the machine. In order to do so, a one-dimensional convolutional autoencoder will be used.

2.2.1 Autoencoder

An autoencoder neural network is an unsupervised learning algorithm that uses a loss function that is optimal when setting the target values to be equal to the inputs, $y_i = x_i$. That is, the neural network attempts to copy its input to its output. Autoencoders consist of three main parts [24], which are shown in Figure 2.3:

1. **Encoder:** this part of the model learns how to map the input state to a reduced space which contains the encoded representation of the data.

2. **Bottleneck:** It is the layer that contains the compressed representation of the input data. This is the lowest- dimensions representation of the input data.

3. **Decoder:** This part of the model learns how to reconstruct the data from the encoded representation to be as close to the original input as possible.

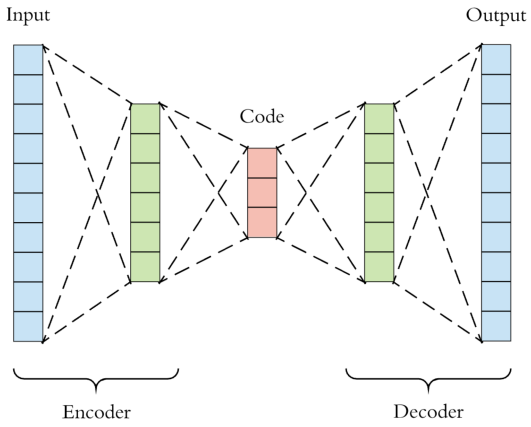


FIGURE 2.3: Architecture of an autoencoder. Figure from [25]

Mathematically, the structure of the autoencoder can be described as:

$$\text{Encoder: } h := f(x)$$
$$\text{Decoder: } \hat{x} := g(h) = g(f(x))$$

where f is a function that represents the encoder layers, x is the input, g is a function representing the decoder layers, and $\hat{x}$ is the output of the autoencoder, an approximation of x.

Autoencoders are usually trained so that the output of the model is as close as possible to the original input of the model. For this reason, the training process consists of minimizing a reconstruction loss, $\mathcal{L}(x, \hat{x})$. This loss is usually the mean square error or the mean absolute error.

There are many applications to autoencoders, here we describe a couple of them.

- **Autoencoders for anomaly detection.** [26] Autoencoders can be used to detect outliers or anomalous data. The autoencoder is trained with a data source until it learns to reproduce its samples. Then, if a new sample belonging to the same data source is provided to the autoencoder, the reconstruction error will be low. On the other hand, if a very different sample (an outlier or anomaly) is fed to the model, the reconstruction error will be high, since the model will fail at reconstructing this sample. In this way, the reconstruction error can be used to detect outliers or anomalous data.

- **Data denoising.** [24] Denoising or noise reduction is the process of removing noise from a signal. The denoising autoencoder tries to minimize the loss $\mathcal{L}(x, g(f(\tilde{x})))$, where $\tilde{x}$ is a noisy input and x is the noiseless input. By minimizing this loss function, the autoencoder has to implicitly learn the structure of the data and undo the corruption, rather than to simply copy the input.

Autoencoder are often designed shallow, i.e. only consisting of a single layer encoder and a single layer decoder. However, training deep architectures having multiple hidden layers can result in a better compression of the data. As deep autoencoders are usually more difficult to train, a common strategy is to pre-train shallow (i.e. with one hidden layer) autoencoders and stack them together. The resulting deep autoencoder can then be trained again based on the already pre-trained layers.

The network architecture for autoencoders can vary depending on the use case. Examples of architectures are simple fully-connected networks, LSTM networks for sequential data or Convolutional Neural Network for image data (2D) or time series (1D).

2.2.2 Anomaly detection

In the previous section, we have seen that one of the applications of autoencoders is anomaly detection. In this project, we will use autoencoders to differentiate between healthy (normal) and unhealthy (anomalous) bearing signals. The autoencoder will be trained only with healthy signals so that the model learns the internal structure of such data. Then, when the model is fed with new healthy data, we expect to have a low reconstruction error. On the other hand, when the model receives an unhealthy sample, it will not be able to reproduce it correctly, since it would have never seen that behaviour before. Therefore, the reconstruction error will be higher, and we will be able to detect the unhealthy bearing state. The reconstruction error is defined as the mean absolute error (MAE) of the reconstructed signal:

$$\text{MAE} = |x - \hat{x}| \tag{2.1}$$

where x is the input sample and $\hat{x}$ is the reconstructed sample.

In order to detect unhealthy data, we have to define a threshold for the reconstruction error. Then, if the reconstruction error is above that threshold, the signal will be considered to be unhealthy. On the other hand, if the reconstruction error is below the threshold, the signal will be considered as being healthy. To define this threshold, we will study the probability distribution of the mean absolute error (MAE). As we will see in Chapter 4, the MAE of the validation set can be modelled using a normal distribution $\mathcal{N}(\mu, \sigma)$. To prove this hypothesis we will perform two normality tests: the Shapiro-Wil test [27] and D'Agostino's K^2 test [28]. Then, a percentile p will be used to set the threshold. That is, if the probability of having a value y of the MAE is less than $1 - p$ (according to the MAE distribution $\mathcal{N}(\mu, \sigma)$), the data will be considered to be unhealthy. Hence, the threshold T will be:

$$T = q(p|\mu, \sigma) \tag{2.2}$$

where q is the quantile function of the normal distribution with mean μ and standard deviation σ. The value of the percentile p is a hyperparameter of the model.

2.3 Classification of the failures

The autoencoder model from the previous section allows to obtain the labels of the bearing failure modes. Having the labelled data set, we will focus on the core of this project: a classification model for bearing failures. The model has to be robust to different operating conditions, such as different machine loads or different rotation velocities. Moreover, the model must not use characteristic information about the machine, such as the rotation velocity or the characteristic bearing failing coefficients. Finally, the training data is limited, and thus the model should be suitable for small data sets. For these reasons, a Deep Learning approach will be selected.

The model consists of a Convolutional Neural Network divided into two parts:

- **The embedding:** It extracts valuable features from the training data, which will be then fed to the classifier. The embedding consists of multiple convolutional layers followed by max-pooling layers. More implementation details can be found in Chapter 3. To train the embedding network, two strategies will be explored: **Siamese networks** and **Triplet Learning**. Such methodologies ensure that the extracted features are robust to different operating conditions. Moreover, triplet learning techniques are especially suitable for situations with very few data samples, which was the other model requirement.

- **The classifier:** It takes as input the features extracted from the embedding and returns the type of fault of the bearing. It consists of multiple fully connected layers. The classifier is trained using cross-entropy loss. Again, for implementation details see Chapter 3.

In the following sections, Siamese networks and triplet learning strategies are introduced and discussed.

2.3.1 Siamese network

The goal of the Siamese network is to create useful embeddings for data points. In our case, the network will learn to map a waveform signal to a point in a lower-dimensional space, while trying to minimize the distance between points of the same class and maximize the distance between points of different classes.

Network structure

Let X_1 and X_2 be two waveform samples. These samples are fed to the feature network G_W, resulting with the feature vectors $G_W(X_1)$ and $G_W(X_2)$, as illustrated in Figure 2.4. The final layer computes pair-wise distance between computed features $E_W = ||G_W(X_1) - G_W(X_2)||_1$ [21]. The label Y of the pair (X_1, X_2) is $Y = 1$ if the two input waveforms are from the same class, and $Y = 0$ otherwise.

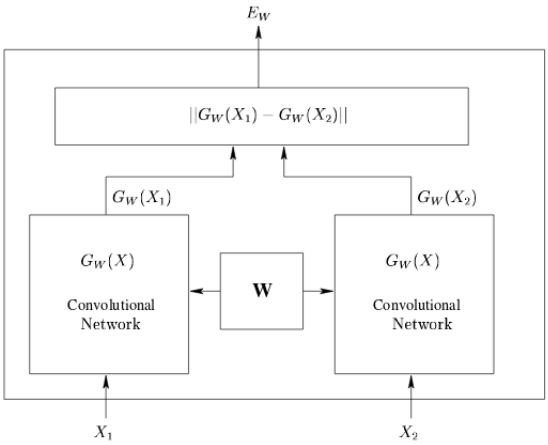


FIGURE 2.4: Architecture of a Siamese network. Figure from [26].

The goal is to train the neural network G_W such that E_W is small if X_1 and X_2 belong to the same category ($Y = 1$), and large if they belong to different categories ($Y = 0$). The system is trained on pairs of samples (X_1, X_2) taken from a training set. Note that the same neural network G_W (with the same weights) is used for the

two members of the pair, X_1 and X_2. For this reason, the network is called *siamese network*.

Contrastive loss

The goal is to define a loss function $\mathcal{L}$ so that by minimizing $\mathcal{L}$ the following properties hold:

- $E_W(X_1, X_2)$ is minimized when X_1 and X_2 are from the same class.
- $E_W(X_1, X_2)$ is maximized when they belong to different classes.

At first glance, we might think that simply minimizing $E_W(X_1, X_2)$ averaged over a set of pairs of inputs from the same class would be sufficient. But this generally leads to a catastrophic collapse: The loss function $\mathcal{L}$ can be made zero by simply making $E_W(X_1, X_2)$ a constant function. Therefore, the loss function needs a contrastive term to ensure not only that the distance E_W for a pair of inputs from the same class is low, but also that the distance E_W for a pair from different classes is large. The loss function is thus of the form [29]:

$$\begin{aligned} \mathcal{L}(W) &= \sum_{i=1}^{N} \mathcal{L}(W, (Y, X_1, X_2)^i), \\ \mathcal{L}(W, (Y, X_1, X_2)^i) &= (1 - Y) \cdot \mathcal{L}_S(E_W(X_1, X_2)^i) + Y \cdot \mathcal{L}_D(E_W(X_1, X_2)^i) \end{aligned} \tag{2.3}$$

where $(Y, X_1, X_2)^i$ is the *i-th* sample, composed by the pair of data samples and the label (same or different class), $\mathcal{L}_S$ is the partial loss function for a "same-class" pair and $\mathcal{L}_D$ is the partial loss function for a "different-class" pair, and N is the number of samples. $\mathcal{L}_S$ and $\mathcal{L}_D$ should be designed in such a way that the minimization of $\mathcal{L}$ decreases $E_W(X_1, X_2)$ for pairs belonging to the same class and increases $E_W(X_1, X_2)$ for different-class pairs. More formally, the following condition must be satisfied:

Property 1. $\exists m > 0$ such that $E_W(X_1, X_2) + m < E_W(X_1, X_2')$, where $Y(X_1, X_2) = 1$ and $Y(X_1, X_2') = 0$.

A simple way to achieve property 1 is to make $\mathcal{L}_S$ monotonically increasing with E_W, and $\mathcal{L}_D$ monotonically decreasing with E_W. More general conditions for $\mathcal{L}_S$ and $\mathcal{L}_D$ can be found in the original paper [29]. Tanking into account the previous conditions, we can define $\mathcal{L}_S$ and $\mathcal{L}_D$:

$$\begin{aligned} \mathcal{L}_S &= \frac{1}{2} E_W^2 \\ \mathcal{L}_D &= \frac{1}{2} \{\max(0, 1 - E_W)\}^2 \end{aligned} \tag{2.4}$$

which satisfies property 1. Finally, the **contrastive loss function** is defined as follows [30]:

$$\begin{aligned} \mathcal{L}(W) &= \sum_{i=1}^{N} \mathcal{L}(W, (Y, X_1, X_2)^i), \\ \mathcal{L}(W, (Y, X_1, X_2)^i) &= (1 - Y) \cdot \frac{1}{2} E_W^2 + Y \cdot \frac{1}{2} \{\max(0, 1 - E_W)\}^2 \end{aligned} \tag{2.5}$$

2.3.2 Triplet strategies

Just as the Siamese network, triplet strategies are also designed to create useful embeddings for data samples. We will see in the following section that there are some advantages over Siamese networks. We first present the formalism of triplet strategies.

Formally, the goal of the triplet strategies [31] is to make sure that:

- Two examples with the **same label** (same failure) have their embeddings **close together** in the embedding space.
- Two examples with **different labels** (different faults) have their embeddings **far away**.

The way to ensure so is by requiring that, given two examples from the same class (positive) and an example from a different class (negative), this negative example should be farther away than the two positive examples by a **margin**. [31]

To formalise this requirement, the loss will be defined over triplets containing [32]:

- an **anchor**.
- a **positive** of the same class as the anchor.
- a **negative** of a different class as the anchor.

Let G_W be a neural network with weights W, which is responsible of producing the waveform embeddings. Let a, p, n be an anchor, a positive and a negative sample respectively. For some distance on the embedding space d (usually the euclidean distance), the loss of a triplet (a, p, n) is:

$$\mathcal{L}(a, p, n) = \max(d(G_W(a), G_W(p)) - d(G_W(a), G_W(n)) + margin, 0) \tag{2.6}$$

By minimizing the loss $\mathcal{L}$, we push $d(G_W(a), G_W(n))$ to be greater than $d(G_W(a), G_W(p))$ by a margin $margin$. When this condition is satisfied the loss function becomes zero. The total loss function is then [33]:

$$\mathcal{L}(W) = \sum_{\substack{a,p,n \\ Y_a = Y_p \neq Y_n}} max(d(G_W(a), G_W(p)) - d(G_W(a), G_W(n)) + margin, 0) \tag{2.7}$$

If this loss is optimized over the whole dataset for long enough, eventually all possible positive pairs (a, p) will be seen and be pulled together and separate from the negative samples. An inconvenience of the triplet loss, though, is that as the dataset gets larger, the possible number of triplets grows cubically, leading to long training times. Therefore, some strategies to select the appropriate triplets are needed.

Triplet mining

Based on the definition of the loss, there are three categories of triplets [32]:

1. **easy triplets:** triplets which have a loss equals to zero, because

$$d(G_W(a), G_W(p)) + margin < d(G_W(a), G_W(n))$$

2. **hard triplets:** triplets where the negative is closer to the anchor than the positive:

$$d(G_W(a), G_W(n)) < d(G_W(a), G_W(p))$$

3. **semi-hard triplets:** triplets where the negative is not closer to the anchor than the positive, but still have positive loss:

$$d(G_W(a), G_W(p)) < d(G_W(a), G_W(n)) < d(G_W(a), G_W(p)) + margin$$

The loss function for easy triplets is already zero, so these triplets do not help to train the embedding network. Moreover, if the triplets are fed to the neural network randomly, it learns relatively quickly to correctly map most trivial triplets, creating a large fraction of uninformative (easy) triplets. Thus, "mining" *hard* triplets is crucial for effective learning [33]. On the other hand, being shown only the hardest triplets would select outliers in the data disproportionally often and make the network G_W unable to learn "normal" associations. Therefore, choosing good triplets to train the network is crucial to achieve a good performance. There are two basic strategies to mine triplets [32]:

1. **Offline triplet mining:**
   With this approach, we compute all the triplets at the beginning of an epoch. We first compute all the embeddings of the training set and only train on the network with hard or semi-hard triplets.

   Overall this technique is not very efficient since we need to do a full pass on the training set to generate triplets. This strategy is similar to the Siamese network since it also computes the pairs of samples before training.

2. **Online triplet mining:**
   The other strategy is to select the triplets for every batch, and train the network with those triplets. This technique gives you more triplets for a single batch of inputs, and does not require any offline mining. It is therefore much more efficient. There are two basic strategies in online triplet mining:

   - **Batch hard strategy**: for each anchor, select the hardest positive (biggest distance $d(G_W(a), G_W(p))$) and the hardest negative among the batch. More formally, consider a batch formed by K classes (types of failure modes) and P samples from each class, resulting in a batch of KP samples. Now, for each sample a in the batch, we can select the hardest positive and the hardest negative samples within the batch when forming the triplets for computing the loss. That is,

$$\mathcal{L}_{Hard}(W) = \sum_{i=1}^{K}\sum_{a=1}^{P} \max(\max_{p=1,\cdots,P} d(G_W(a^i), G_W(p^i))$$
$$- \min_{\substack{j=1,\cdots,K \\ n=1,\cdots,P \\ j\neq i}} d(G_W(a^i), G_W(n^i)) + margin, 0)$$

Where the first term corresponds to the hardest positive and the second term corresponds to the hardest negative. The super index i refers to the *i-th* sample. With this definition, we obtain PK terms contributing to the loss, which is about three times over the traditional formulation [33]. Additionally, the selected triplets can be considered moderate triplets since they are the hardest within a small subset of the data, which is exactly what is best for learning with the triplet loss.

- **Batch all strategy**: select all the valid triplets (triplets containing an anchor, a positive and a negative sample), and average the loss on the hard and semi-hard triplets. That is to, simply use all possible $PK(PK-P)(P-1)$ combinations of triplets.

$$\mathcal{L}_{All}(W) = \sum_{i=1}^{k}\sum_{a=1}^{P}\sum_{\substack{p=1\\p\neq a}}^{P}\sum_{\substack{j=1\\j\neq i}}^{K}\sum_{n=1}^{P} max(d(G_W(a^i), G_W(p^i)) - d(G_W(a^i), G_W(n^j)) + margin, 0) \quad (2.8)$$

The batch all strategy is specially suitable when the number of data samples is small, since it produces many more samples that the ones in the training data. On the other hand, when the data set is large enough the batch hard strategy has proven to have very good results. Figure 2.5 shows a representation of both online and offline triplet mining.

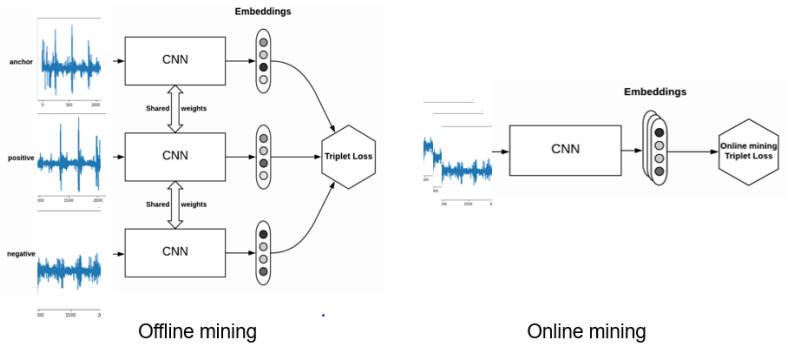


FIGURE 2.5: Comparison of offline and online triplet learning. Figures are based on [32].

2.3.3 Comparison of both methods

So far we have seen two methodologies to create useful embeddings for data samples: Siamese network and triplet learning. Both methods aim to generate embeddings which are robust to different operating conditions, such as multiple rotation velocities. They do so by placing the representation of the samples close together in the embedding space if and only if the failure mode is the same, regardless of the rotation velocity. There are, however, a couple of advantages in choosing the triplet learning approach:

- With triplet learning strategies, only informative samples are fed to the network. That is, the easy samples which already fulfil the desired condition in

the embedding space are not used to train the network. Therefore, training is more efficient.

- Triplet learning is very useful for small data sets, especially with the batch all strategy. The number of triplets used to train the network grows cubically with the number of samples, instead of quadratically as in the Siamese network approach.

The main disadvantage of the triplet learning approach is that it is harder to implement, especially if you add the constraints of building a computational graph in TensorFlow.

2.4 Continuous learning

When dealing with machinery problems, usually all the historical data is not available during the training stage. The machine is constantly working and slowly evolving, hence new data arrives progressively with time. It is thus important to continuously train the model with time. Sometimes, this new data can contain unseen behaviour to the model. The change of the probability distribution of the input data is known as *concept drift* [34]. This change may be due to a *hidden context*, unknown to the model. For example, if the machine operates near to another functioning machine, this second machine can affect the vibration signal of the first one, and thus produce a change in the data. If these machines only are working at the same time some specific days, we will want the model to learn to correctly detect failures both when the machines are working together and when they are not.

When a concept drift occurs the challenge for a neural network is to learn the new data (data after concept drift: task B) while simultaneously maintaining the performance of the old data (data before concept drift: task A). If the data from both tasks are available at the same time, the best option is to train the model mixing both data sets. Even if this approach can lead to the best performance, it has two disadvantages. First, all the data has to be stored in a database, which may require a lot of storage space. Moreover, if the training data sets from both tasks are large, training a model with both of them may be computationally inefficient. Another option could be training the network for the first task and then for the second task. However, sequential learning models tend to forget previously learnt task, a phenomenon termed as catastrophic forgetting [35]. The following section introduces an algorithm which tries to overcome this phenomenon.

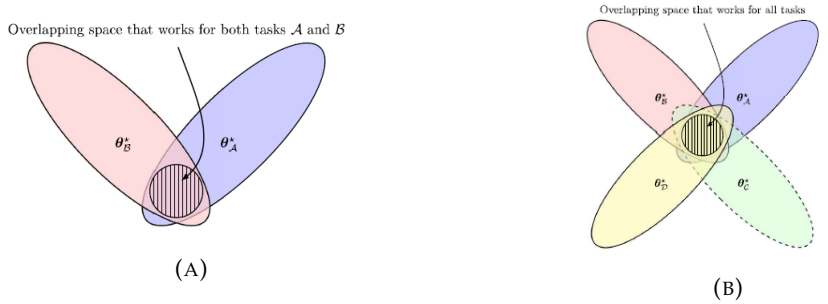
2.4.1 Elastic weight consolidation

The idea behind Elastic weight consolidation (EWC) [35] comes from the way the mammals' brain works. That is, learning new tasks and avoiding to erase the synaptic connections that are most useful to perform other tasks.

Denote by θ the weights of a certain neural network. Training the network means finding the optimal parameters θ^* which minimizes the error of the model output. The key point is that due to the high complexity of neural network models, there may be multiple configuration of weights and biases that produce an optimal performance (within an acceptable error). The goal then is to find a value of the network parameters that are optimal for all the tasks that the model aims to accomplish. We will begin by describing how to learn two tasks: A and B.

Learning a second task after the first

Imagine a certain model has been trained to perform task A, having learnt the optimal parameters θ^*_A. The idea is to find a configuration of parameters $\theta^*_{A,B}$ that performs well both tasks. In order to do so, the parameters must be chosen from the intersection of the solution spaces of tasks A and B, as seen in Figure 2.6.



(A) (B)

FIGURE 2.6: Example of the solution spaces for two tasks (A) and for four tasks (B). Figures from [36].

The basic idea behind EWC is to restrict the weights which are important for the execution of task A when learning task B. In the algorithm this restriction is implemented as a weighted L_2 regularization. The question is how to identify the weights that are of great importance for task A.

Let D_A be the data from task A and D_B be the data from task B. To formulate the loss function, we start by taking a Bayesian approach to estimate the parameters of the network θ.

$$p(\theta|D_A, D_B) = \frac{p(D_B|D_A, \theta)p(\theta|D_A)}{p(D_B|D_A)} \tag{2.9}$$

where $p(\theta|D_A, D_B)$ is the posterior probability of θ, $p(\theta|D_B)$ is the likelihood and $p(\theta|D_A)$ is the prior probability. Usually it is easier to maximize the logarithm of a probability, because of its additive property. In this case, we formulate the loss function of learning tasks A and B as the negative log-posterior probability of both tasks. Notice that this is a different approach than maximizing the log-likelihood of both tasks. Instead of looking for the maximum likelihood estimator of tasks A and B, we look for the *maximum a posteriori* estimator, which is easier to approximate in this case. The posterior log probability of the parameters is given by:

$$\begin{aligned}\log p(\theta|D_A, D_B) &= \log p(D_B|D_A, \theta) + \log p(\theta|D_A) - \log p(D_B|D_A) \\ &= \log p(D_B|\theta) + \log p(\theta|D_A) - \log p(D_B|D_A)\end{aligned}$$

The left-hand side is the posterior probability of the final model (which in this case is the negative of the loss function we want to minimize). The term $-\log p(D_B|\theta)$ is the loss function for task B, and the term $\log p(D_B|D_A)$ does not depend on θ and therefore can be treated as a constant. The last term which needs to be calculated is $\log p(\theta|D_A)$, which can be thought as an adaptive elastic weight regularizer that tries to maintain knowledge of task A [37]. However, this term is intractable, in the sense that it is difficult to find a closed-form. For this reason, this term will be approximated using the diagonalised Laplace approximation.

Diagonalized Laplace approximation [36]

As we have already mentioned, we aim to approximate the posterior probability $p(\theta|D_A)$ of task A. The basic idea of the Laplace approximation is to find a normal distribution approximation to a continuous probability density distribution. Under the condition that $p(\theta|D_A)$ is smooth and peaked around its maximum θ^*_A, the posterior probability distribution can be approximated by a normal distribution with mean θ^*_A and variance $\mathbb{1}_A^{-1}$ [36].

In order to prove this last statement, let us compute the Taylor expansion of the loss function $\mathcal{L}(\theta) = -log(p(\theta|D_A)) = \mathcal{L}_A(\theta) - log(p(\theta))$ around θ^*_A. Here, $\mathcal{L}_A(\theta)$ is the usual loss function of task A and $p(\theta)$ acts as a regularizer on the parameters of the network. This regularization can be the usual L^2 loss.

$$\mathcal{L}(\theta) = \mathcal{L}(\theta^*_A) + \frac{\partial \mathcal{L}(\theta)}{\partial \theta}\Big|_{\theta^*_A}(\theta - \theta^*_A) + \frac{1}{2}(\theta - \theta^*_A)^T\Big(\frac{\partial^2 \mathcal{L}(\theta)}{\partial \theta^2}\Big|_{\theta^*_A}\Big)(\theta - \theta^*_A) + O(3) \quad (2.10)$$

Since the loss function has its maximum in θ^*_A, we have that $\frac{\partial \mathcal{L}(\theta)}{\partial \theta}\Big|_{\theta^*_A} = 0$. Neglecting higher order terms we obtain:

$$\mathcal{L}(\theta) \approx \mathcal{L}(\theta^*_A) + \frac{1}{2}(\theta - \theta^*_A)^T\Big(\frac{\partial^2 \mathcal{L}(\theta)}{\partial \theta^2}\Big|_{\theta^*_A}\Big)(\theta - \theta^*_A) \quad (2.11)$$

Equation 2.11 is equivalent to

$$log(p(\theta|D_A)) \approx log(p(\theta^*_A|D_A)) + \frac{1}{2}(\theta - \theta^*_A)^T\Big(\frac{\partial^2 log(p(\theta|D_A))}{\partial \theta^2}\Big|_{\theta^*_A}\Big)(\theta - \theta^*_A) \quad (2.12)$$

The term $log(p(\theta^*_A|D_A))$ is constant with θ. Finally, writing

$$\Big(\frac{\partial^2 log(p(\theta|D_A))}{\partial \theta^2}\Big|_{\theta^*_A}\Big) = -\Big(\Big(-\frac{\partial^2 log(p(\theta|D_A))}{\partial \theta^2}\Big|_{\theta^*_A}\Big)^{-1}\Big)^{-1}$$

we obtain that

$$p(\theta|D_A) = \Delta \exp\Big(-\frac{1}{2}(\theta - \theta^*_A)^T\Big(\Big(-\frac{\partial^2 log(p(\theta|D_A))}{\partial \theta^2}\Big|_{\theta^*_A}\Big)^{-1}\Big)^{-1}(\theta - \theta^*_A)\Big) \quad (2.13)$$

where $\Delta = p(\theta^*_A|D_A)$. Therefore, we conclude that using the Laplace approximation, the posterior probability $p(\theta|D_A)$ follows a normal distribution.

$$p(\theta|D_A) \approx \mathcal{N}\Big(\theta^*_A, \Big(-\frac{\partial^2 log(p(\theta|D_A))}{\partial \theta^2}\Big|_{\theta^*_A}\Big)^{-1}\Big) \quad (2.14)$$

The **Fisher information matrix** is defined as

$$\mathbb{1}_A = \mathbb{E}[\Big(-\frac{\partial^2 log(p(\theta|D_A))}{\partial \theta^2}\Big|_{\theta^*_A}\Big)] \quad (2.15)$$

And therefore, the posterior distribution for task A follows approximately a normal distributions with parameters $\mathcal{N}(\theta^*_A, \mathbb{1}_A^{-1})$.

So far we have described the Laplace approximation for $p(\theta|D_A)$. The last step is to further approximate the Fisher information matrix $\mathbb{1}_A$ to gain computational efficiency. Note that $\mathbb{1}_A$ can be calculated using only first derivatives:

$$\begin{aligned}\mathbb{1}_A &= \mathbb{E}[\Big(- \frac{\partial^2 log(p(\theta|D_A))}{\partial\theta^2}\Big|_{\theta_A^*}\Big)] \\ &= \mathbb{E}[\Big(\frac{\partial log(p(\theta|D_A))}{\partial\theta}\Big|_{\theta_A^*}\Big)\Big(\frac{\partial log(p(\theta|D_A))}{\partial\theta}\Big|_{\theta_A^*}\Big)^T]\end{aligned}$$

In practice, the distribution $p(\theta|D_A)$ is unknown, and therefore an empirical approximation of $\mathbb{1}_A$ is computed:

$$\begin{aligned}\hat{\mathbb{1}}_A =& \frac{1}{N}\sum_{i=1}^{N} \nabla log(p(\theta|D_A))\Big|_{\theta_A^*} \nabla log(p(\theta|D_A))\Big|_{\theta_A^{*T}} \\ =& \frac{1}{N}\sum_{i=1}^{N} \nabla\mathcal{L}_A(x_{A_i}|\theta_A^*)\nabla\mathcal{L}_A(x_{A_i}|\theta_A^*)^T + \lambda_{\text{prior}}\mathbb{1}\end{aligned} \tag{2.16}$$

where $\lambda_{\text{prior}}\mathbb{1}$ is the Hessian of the log-prior $log(p(\theta))$, assuming an L^2 regularization. In the original paper, they set $\lambda_{\text{prior}} = 0$. Finally, the $\hat{\mathbb{1}}_A$ matrix is approximated by its diagonal form $[\hat{\mathbb{1}}_A]$.

$$[\hat{\mathbb{1}}_A] = \frac{1}{N}\sum_i \Big(\frac{\partial\mathcal{L}_A(x_{A_i}|\theta_A^*)}{\partial\theta_{A_i}}\Big)^2 + \lambda_{\text{prior}}\mathbb{1} \tag{2.17}$$

Putting it all together we obtain that:

$$\begin{aligned}\log p(\theta|D_A, D_B) =& \log p(\theta|D_B) + \log p(\theta|D_A) - \log p(D_B|D_A) \\ \approx& logp(\theta|D_B) + \frac{1}{2}(\theta - \theta_A^*)^T\mathbb{E}[\Big(\frac{\partial^2 log(p(\theta|D_A))}{\partial\theta^2}\Big|_{\theta_A^*}\Big)](\theta - \theta_A^*) \\ \approx& -\mathcal{L}_B(\theta) - \sum_i([\hat{\mathbb{1}}_A]_i + \lambda_{\text{prior}}\mathbb{1})(\theta_i - \theta_{A_i}^*)^2 + \text{constant}\end{aligned}$$

where $[\hat{\mathbb{1}}_A]_i$ is the *i-th* diagonal component of the empirical Fisher information matrix. Finally, the loss function for learning task B without forgetting about task A is the following.

$$\boxed{\mathcal{L}_{A,B}(\theta) = \mathcal{L}_B(\theta) + \lambda\sum_i([\hat{\mathbb{1}}_A]_i + \lambda_{\text{prior}}\mathbb{1})(\theta_i - \theta_{A_i}^*)^2} \tag{2.18}$$

where λ is a parameter introduced to have a trade-off between learning task B and not forgetting task A.

Learning the third task after the first two

In the previous sections, we have seen how to train a model for task B without forgetting about task A. What happens if we want to learn task C without forgetting about tasks A and B? Note that this will be the case in our setting since the model will be updated when new data arrives and we do not want to forget about all the previously learnt tasks.

In order to learn task C, let's apply the Bayes theorem to $p(\theta|D_A, D_B, D_C)$:

$$log(p(\theta|D_A, D_B, D_C)) = -\mathcal{L}_C + log(p(\theta|D_A, D_B)) + \text{constant} \tag{2.19}$$

However, we are assuming that we have already trained a model for both tasks A and B. Therefore, the optimal weights obtained in the previous step are $\theta^*_{A,B}$ and the data sets D_A, D_B are in reality $D_{A,B}$. Therefore, we recover the case with only two tasks. The final loss function is

$$\boxed{\mathcal{L}_{A,B,C}(\theta) = \mathcal{L}_C(\theta) + \lambda \sum_i ([\hat{\mathbb{1}}_{A,B}]_i + \lambda_{\text{prior}}\mathbb{1})(\theta_i - \theta^*_{A,B_i})^2} \tag{2.20}$$

where $[\hat{\mathbb{1}}_{A,B}]_i$ is the diagonalized empirical fisher information matrix for tasks A and B.

Error in EWC

In this section we look at the sources of error when applying the diagonalized Laplace approximation to estimate the posterior probability $p(\theta|D_A)$. The goal of this section is not to give a formal proof of the validity of this approach, but to be aware of the all the approximations used in this algorithm and understand where these approximations are appropriate. The sources of error of the EWC are the following:

1. **Laplace approximation**: The Laplace approximation is used to model the posterior probability $p(\theta|D_A)$ around the optimal value θ^*_A with a normal distribution. Since θ^*_A is a local minimum of $p(\theta|D_A)$, the probability distribution has a bell shape around this point, similar to the shape of a normal distribution. This approach uses two main approximations:

   - **Taylor series**: In order to estimate $\log(p(\theta|D_A))$, a Taylor expansion of order two around θ^*_A is used. The error of this approximation is of order $O((\theta - \theta^*_A)^3)$. If the two tasks D_A and D_B are not very different, we expect that the optimal $\theta^*_{A,B}$ of both tasks will be close to θ^*_A and thus the terms of order $O((\theta - \theta^*_A)^3)$ will be small.
   - **Empirical estimation of** $\mathbb{1}_A$: The expected value of the Fisher information matrix is estimated using the real data from data set D_A. By the law of large numbers, the empirical mean value converges to the expected value when the number of samples, N, tends to infinity. Moreover, under the assumption of finite variance (σ^2) and no correlation between random variables, the variance of this average of N random variables is σ^2/N. Therefore, the error of estimate of the expected value of the Fisher information matrix decreases as $\frac{1}{N}$, when $N \longrightarrow \infty$.

   In fact, the Laplace approximation has formally proven to be correct at the limit of large data sets. The Bernstein-von Mises theorem [38] proves this fact when the model is regular, the prior probability density function is smooth and the sample size n is large. Here we present the formulation of the theorem, the prove can be found in [39].

   Theorem 1 (Bernstein-von Mises)**.** *Consider the problem of analysing N IID datapoints $\{X_i\}$, according to some Bayesian model, composed of a log-prior $f_0(\theta)$, $\theta \in \Theta$ a common log-density likelihood $d(X|\theta)$ and a log-posterior density function $f(\theta)$:*

$$f(\theta) = f_0(\theta) + \sum_{i=1}^{N} d(X_i = x_i|\theta) \tag{2.21}$$

Let θ^* be a local maximum of $f(\theta)$. Assume the following properties:

(a) The prior $\exp(f_0(\theta))$ has a non-zero density on all neighborhoods of θ^*.

(b) The posterior probability of the event $||\theta - \theta^*||_2 \leq \epsilon$ converges to 1 as $N \longrightarrow \infty$.

(c) $d(x|\theta)$ is a regular distribution. More precisely:

- *The average function $\theta \longrightarrow \mathbb{E}[d(x|\theta)]$ has a positive Hessian at θ^*.*
- *The gradient of $d(x|\theta)$ at θ^* has finite vaiance.*
- *In a local neighborhood around θ^*, $d(x|\theta)$ is m-Lipschitz where m is a random variable such that $\mathbb{E}[m^2]$ is finite.*

Then, the total-variation distance between the posterior probability $f(\theta)$ and its Laplace approximation $g_{LAP}(\theta)$ converges to zero as:

$$d_{TV}(f(\theta), g_{LAP}(\theta)) = O(\frac{1}{\sqrt{N}}) \tag{2.22}$$

Where

$$d_{TV}(f(\theta), g_{LAP}(\theta)) = \sup_{\theta \in \Theta} |f(\theta) - g_{LAP}(\theta)| \tag{2.23}$$

Therefore, if the optimal value of task A, θ_A^* is correctly estimated and the likelihood of the model is regular, the Laplace approximation is asymptotically correct.

2. **Diagonal Fisher matrix**: Finally, for computation efficiency, the Fisher information matrix (FIM) is approximated by its diagonal form. Although there are not clear error bounds for this approximation, if the matrix is diagonally-dominant, the approximated matrix contains the most relevant terms of the original matrix. Since the FIM is the Hessian matrix of $f(\theta) = -log(p(\theta|D_A))$ around θ_A^*, and $f(\theta)$ is concave around its maximum θ_A^*, the FIM is positive definite and therefore diagonally-dominant.

Chapter 3

Implementation details

This chapter provides the implementation details of all the models used in this project. First, it presents the different data sets and the preprocessing techniques used in each of them. Then, the model architectures are presented. Furthermore, to make the project accessible to an end-user, the algorithms are wrapped up in a web application. This web application is presented at the end of this chapter.

3.1 Data sets

In this project, we use two data sets of bearing vibration signals. This data sets are sometimes mixed up to simulate multiple operating conditions.

3.1.1 IMS data set

The first data set is provided by the NSF I/UCR Center for Intelligent Maintenance Systems (IMS) in collaboration with Rexnord Corp. and publicly available [40]. For the tests, four Rexnord ZA-2115 double row bearings were installed on a shaft. The shaft was set in motion by an AC motor with a radial load of 6000 lbs with a spring mechanism. The rotation speed was kept constant at 2000 rpm or 33.33 Hz. The vibration signal was captured by a PCB 353B33 High Sensitivity Quartz ICP accelerometer installed on the bearing housing.

This data set contains the time evolution of different bearings. At the end of the tests, some of these bearings suffer from failures. In total the experiment consists of three independent test-to-failure experiments with four bearings for each experiment. Each data set gives a 1-second vibration signal snapshots recorded at specific intervals which are usually every 10 minutes. Each 1-second snapshot consists of 20480 data points or a sample frequency of 20.48 kHz. The occurred failures are given in the table

	number of files	Duration	Damage occurred
Test 1	2156	49680 min	Bearing 3: Inner ring Bearing 4: Rolling element
Test 2	984	9840	Bearing 1: outer ring
Test 3	4448	44480	Bearing 3: outer ring

TABLE 3.1: Description of the IMS data set.

3.1.2 Paderborn data set

This data set contains vibration data (waveforms) of different bearings, each suffering from a fault (inner ring, outer ring) or being healthy. The fault is completely developed. This data set is provided by Paderborn University, and it is publicly available at [41]. It contains bearings working at different operating conditions. The rotational speed of the drive system, the radial force onto the test bearing and the load torque in the drive train are the main operating parameters. All three parameters were kept constant for each measurement. The table below shows the operating condition of each asset.

Nº	Rotational speed [rpm]	Torque [Nm]	Radial force [N]	Name
0	1500	0.7	1000	N15_M07_F10
1	900	0.7	1000	N09_M07_F10
2	1500	0.1	1000	N15_M01_F10
23	1500	0.7	400	N15_M07_F04

TABLE 3.2: Description of the Paderborn data operating conditions.

3.2 Preprocessing techniques

Preprocessing techniques are applied for two main reasons:

- Make the data more similar to real data. Since the data sets we are using have been simulated artificially, the vibration signal we obtain from them may be different from real data. Therefore, it is interesting to transform the data in such a way that it resembles data from real clients.
- Prepare the data to feed the neural network models.

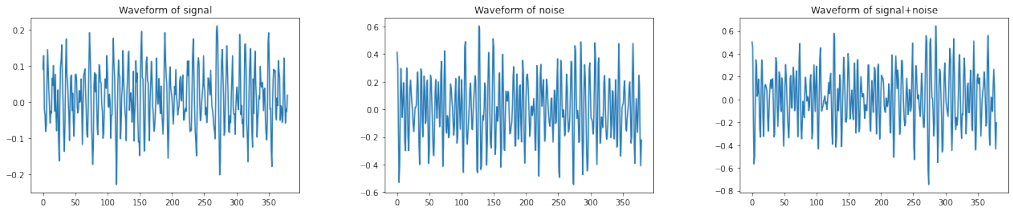
The preprocessing techniques used in this project are the following:

- Reduce the resolution by downsampling the sample frequency to 5kHz. This is the usual sampling frequency in the business samples.
- Use a sliding window to crop the waveform signal to a certain number of time stamps. This is done to feed the model with a fixed input size. The number of time stamps should be large enough to contain at least two time periods of the rotating frequency of the machine [2]. On the other hand, a too large input size for the neural network can create difficulties during training. Therefore, we treated this value as a hyperparameter and chose the value which was more suitable according to our experiments. At the end, we selected an input size of 4000 timestamps for the autoencoder and 1000 timestamps for the classification model.
- The data was standardized and rescaled into the value range of $[0, 1]$. Not only does this lead to an improved convergence of the models but it furthermore forbids the models of just learning the amplitude of the waveform.

Paderborn data set turns out to be noiseless, which is unrealistic compared with real data. Moreover, the different waveforms are very similar because the failure was completely developed at the beginning of the experiment. In order to solve these issues, noise is added to the original waveforms. The noise is introduced in two different ways:

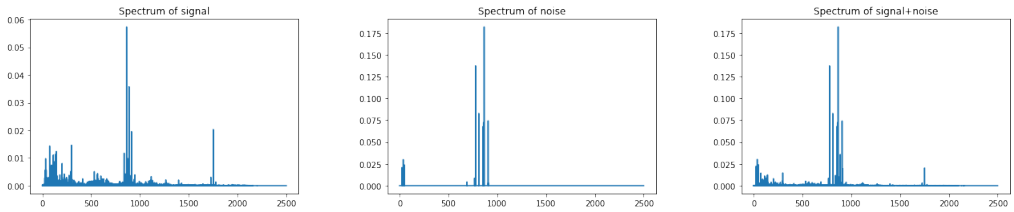
1. Amplify the amplitude of the 4rth first harmonics.
2. Introduce white noise (random frequencies) within a frequency band of 750-1000Hz.

The following figures give an example to show what the noise looks like, both in the time (waveform, Figure 3.1) and frequency (spectrum, Figure 3.2) domain.



(A) (B) (C)

FIGURE 3.1: Illustration of the waveform signal and the added noise.



(A) (B) (C)

FIGURE 3.2: Illustration of the spectrum signal and the added noise.

3.3 Models architecture

In this section, we present the architecture of all the models, as well as the learning strategies. The process of selecting the model architecture is the following. We start with a low-complexity model, that is, with a low number of layers and parameters. This model will produce low-performance results in both train and validation sets. Then, we increase the complexity of the model by increasing the number of layers and the number of parameters in each layer, until the performance on the training set is higher than the performance on the validation set. At that point, the complexity is too high and the model has started to produce overfitting. Then, we decrease the complexity until the performance on both the training and validation sets is similar. We stay with that model architecture. We follow this strategy for all the models.

3.3.1 Autoencoder

In Chapter 2 we talked about the importance of detecting the bearing failure at early stages of its development. In order to do so, labelled data containing the full development of the failure is needed. In order to estimate the beginning of a failure, we propose a one-dimensional convolutional autoencoder (1D CNN AE). The idea behind using a 1D CNN AE is that the convolutional layer extract the essential features such that the model is able to properly learn the data generating process without prior feature creation.

As the training dataset is usually quite small it is important to keep the number of trainable parameters as small as possible. In the model at hand, this was achieved by avoiding fully connected layers. Recall that the autoencoder is composed of two models: the encoder and the decoder. The encoder CNN has four 1D Convolutional layers with 128, 256, 512 and 6 filters, height = 30, 20, 10, 5, strides=1 and padding='same'. We have three max-pooling subsampling layers with pool size = 2. All layers have Relu activation function. The output of the Encoder CNN is the bottleneck, which is the input of the decoder. The decoder is the opposite version of the encoder. It has 3 convolutional layers with 512, 256 and 1 filters respectively. These are mixed with upsampling layers of size=2. Also, two drop out layers are introduced after the first two max-pooling layers, to reduce the chances of overfitting.

The model is implemented using Tensorflow 2. The model definition is placed inside a class Autoencoder, which extends the *Model* class from *tf.keras*. The training methods are written inside a class called *Training_AE*. This class contains the fit method, as well as an early stopping method and a method for checkpointing the model, i.e. saving the weights, optimizer state etc.

After training the model, the threshold for the reconstruction error is calculated. In order to do so, we fit the error distribution with a Gaussian distribution $\mathcal{N}(\mu, \sigma)$. Both the Shapiro-Wilk test and the D'Agostino's K^2 test are performed to test the normality hypothesis. If accepted, the mean and variance of such distribution will be calculated. Finally, a percentile p is chosen to determine the threshold. This percentile is a hyperparameter of the model. We choose p so that most of the healthy data is below the threshold whilst allowing early failure detection.

3.3.2 Siamese network

After obtaining the data labels from the autoencoder model, a classification model for bearing failures is designed. For the first approach, we use a Siamese network, which is described in Chapter 2.

In order to design a Siamese network, we use a 1D convolutional neural network as the embedding network. Then, some fully connected layers are added to classify the bearing faults based on the embeddings extracted by the CNN. The CNN has three 1D Convolutional layers with 32 filters, height=10, strides=1 and padding='same'. After the convolutional layers, there are 3 MaxPooling subsampling layers with pool size = 20, strides = 4. Then we have 1 FC with 256 nodes, one with 128 nodes, and another with 20 output nodes. All other layers have Relu activation function. This network extracts the embeddings. The classification layer is a softmax layer with 4 outputs. The training process is done as follows.

1. **Pretraining:** In the first step, the whole classifier is trained using classification cross-entropy loss. This step is done to obtain a good initialisation of the embeddings. This step makes the optimization process easier because at the end of the pretraining phase we already have a good approximation of the optimal result.

2. **Training the embeddings and the classifier:** After pre-training the network, the embeddings are trained using the contrastive loss, and the classifier is trained using the cross-entropy loss. We add a hyperparameter w, which is the weight of the contrastive loss with respect to the cross-entropy loss. We

found out that setting $w = 10$ provides a good balance between the two loss functions.

Both the pretraining and the training functions are wrapped up in a class *Bearings_Network*. This class contains the fit method, as well as an early stopping method and a method for checkpointing the model.

3.3.3 Triplet learning network

Triplet learning strategies is the second approach we tested to design a classification model of bearing failures.

Similarly to the Siamese network, we use a 1D convolutional neural network to generate the embeddings. Then, some fully connected layers classify the failure based on the output of the embedding. The embedding layers consist of four 1D Convolutional layers with 32, 32, 32 and 64 filters respectively, height=10, strides=4 and padding='same'. After each convolutional layer, a max-pooling subsampling layer with pool size = 10, strides = 4 is added. Afterwards, we have two fully connected layers with 32 nodes and 20 output nodes. This last layer is the output of the embedding. All layers have Relu activation function. The classification model takes as input the output of the embedding model, and adds three fully connected layers, with 64, 64 and 32 filters each. The output is a dense layer with 4 outputs. The training process is done as follows:

1. **Pretraining:** Again, we start pretraining the whole network using a cross-entropy loss, to obtain a good initialization of the embeddings.
2. **Training the embeddings:** The next step is to train the embeddings using the triplet loss. We implemented the two online triplet strategies: batch all and batch hard. In this step, it is important to use low learning rates. Otherwise, the network may collapse to a single point (since it is a local minimum of the loss function), obtaining a loss value equal to the margin. Moreover, a good initialization of the weights is useful to overcome this issue.
3. **Training the classifier:** Once the embeddings have been trained, we train the classifier layers using the cross-entropy loss. We have separated the training of the embeddings and the classifier since we use different values of learning rates and training epochs.

The model definitions are found in the classes *Embedding* and *classifier* respectively. Then we define three classes for the training process: *Pretraining*, *Train_Embeddings* and *Training_classifier*. All these classes contain fit methods and checkpointing methods to save and restore the models.

3.3.4 Elastic weight consolidation

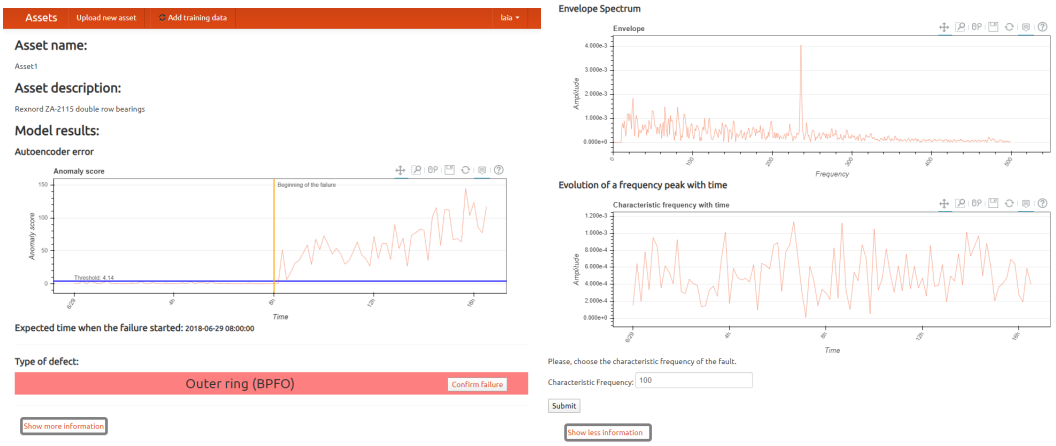
To make experiments using elastic weight consolidation, a classification model is first trained for one data set. Then, using EWC methodology, the network is trained to learn to classify failures from another data set. The training of the first model is done just as we described in the section above. Then, a class called *Fisher_matrix* is used to create and store the Fisher information matrix for both the embedding network and the classifier. Then, using EWC methodology, the embedding network is trained with fisher loss and EWC loss. The parameter λ is chosen so that both

triplet and EWC loss functions have similar magnitudes. The learning rate is set to be small so that the optimization does not converge to optimizing only one of the losses. Finally, the classifier is trained using cross-entropy loss and EWC loss. The EWC training processes are defined in the classes *Train_Embeddings_EWC* and *Train_Classifier_EWC*.

3.4 Web application

The objective of the web application is to provide a bearing fault monitoring tool which is easy to handle for the end-user. The designed app has to provide a database to store the streaming data, incorporating the retraining of the models and provide a visualization of the results. The application was written in Flask. The database is chosen to be MongoDB and the plots are designed in Bokeh. The application was developed with a colleague at Accenture.

Figure 3.3 shows the main page of the application. In the upper part, the autoencoder reconstruction error with the threshold are depicted indicating an error in the last part of the graph. The type of defect is the result of a classification model. The part in the bottom of the web page provides the user with the opportunity to inspect the envelope spectrum. There are two different ways to display the envelope spectrum. The first one is to show the envelope spectrum for a given waveform, selected by the user. Additionally, the user can display the evolution of the of a specific frequency over time. If the frequency corresponds to a bearing failure frequency, this figure allows to analyse the bearing failure using vibration analysis.



(A) (B)

FIGURE 3.3: Web application.

Chapter 4

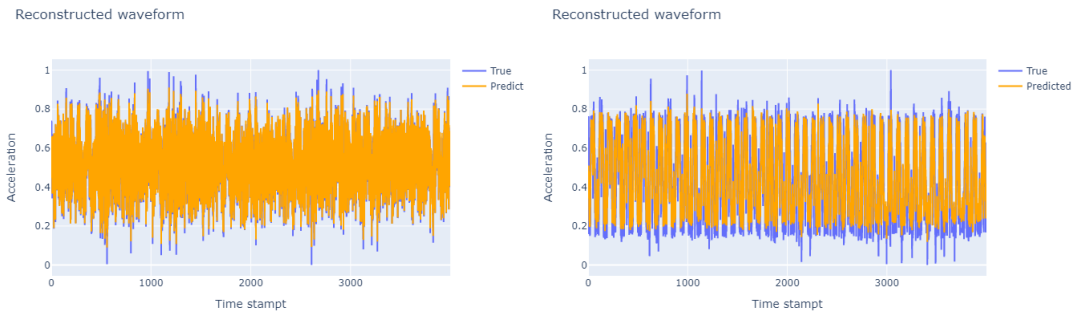
Results

This chapter presents the results from the methodologies explained in the previous chapters.

4.1 Autoencoder

The goal of the autoencoder model is to perform early detection of the bearing failures. This allows to put labels to the bearing failure data, which will be used to develop a classification model.

The model is trained using the IMS data set, which contains the time evolution of bearing elements. At the end of the tests, some of the bearings have acquired a failure. We inspect four different bearings which, at the end of the test, have an inner ring failure, an outer ring failure, a rolling element failure and no failure respectively. The model is trained using the first 500 time stamps of the inner ring bearing. We know that this data is healthy because the inner ring error has a fast development, and the whole experiment has more than 2000 time stamps. Then, the model is tested with all four bearings.



(A) (B)

FIGURE 4.1: Reconstructed and real waveforms. In (A) the waveform is healthy. In (B) the waveform suffers from an outer ring error.

Figure 4.1 shows the superposition of the real and the predicted waveforms for both a healthy and a faulty waveform. It can be seen that when the data is healthy, the autoencoder reconstructs the signal much better than when it is faulty. Therefore, the reconstruction error can be a measure of the bearing health state.

In order to decide when the error development starts, we model the reconstruction error probability distribution. Except for some occasional outliers, the error

follows a normal distribution. This fact was checked by plotting a Q-Qplot of the reconstruction error distribution and performing two normality tests: the Shapiro-Wilk test and the D'Agostino's K^2 test. The p-values for these tests are 0.875 and 0.552. Since both values are higher than 0.05, we can not reject the normality hypothesis with a confidence level of 95%. The parameters of the normal distribution are estimated to be $\mu = 0.0116$ and $\sigma = 0.00164$. Figure 4.2 shows the histogram of the probability distribution of the reconstruction error, together with its normal fit. According to this distribution, the threshold T is estimated to be the quantile with percentile $p = 0.95$. The value of the threshold is

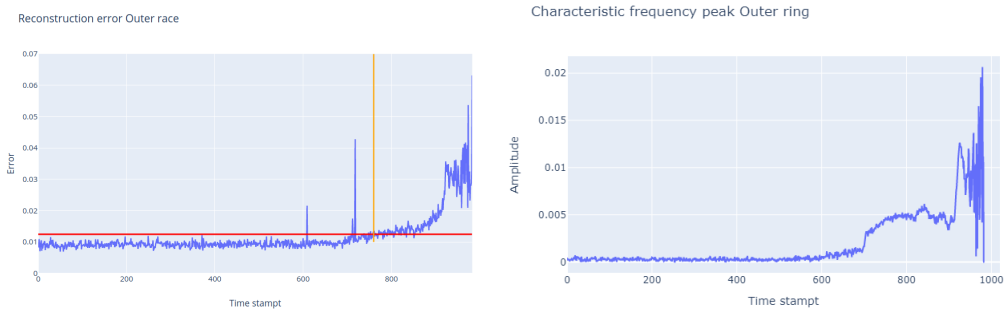
$$T = 0.0125$$

FIGURE 4.2: Probability distribution of the autoencoder reconstruction error.

Now that the threshold has been determined, we inspect the reconstruction error for the four bearings of our study. In order to assess the performance of the autoencoder, we perform a model-based analysis as well and compare both results. In this case, we *do* know the characteristic frequencies of the bearing faults, which is not usually the case when dealing with real data. Therefore, we can use the bearing failing frequencies to inspect the true value of the beginning of the failure. In order to do so, we calculate the spectrum for each waveform. If there is a failure, there should be a pronounced peak in the corresponding bearing failing frequency. By looking at the amplitude of such peak over time, we can infer when the error began its development.

Outer ring failure

Figure 4.3 shows both the evolution of the outer ring bearing failing frequency and the autoencoder reconstruction error. Both figures suggest that the failure starts at around timestamp 760, and becomes severe at timestamp 900. These figures prove that the autoencoder approach performs as good as the model-based approach, and does not use characteristic information of the bearing.

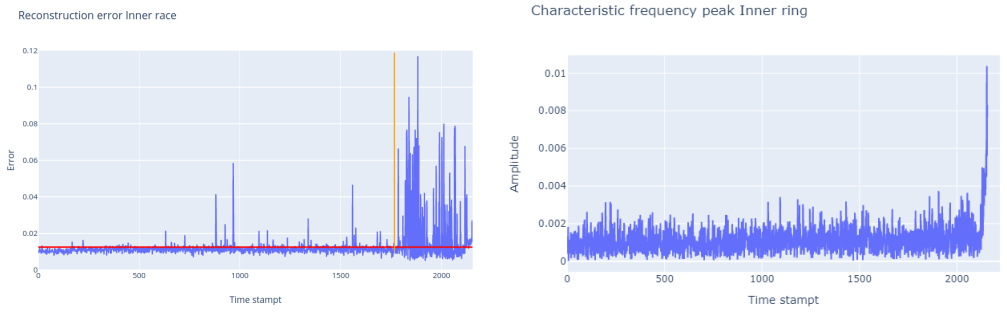


(A) (B)

FIGURE 4.3: Evolution of the outer ring bearing failing frequency (A) and autoencoder error (B) for the outer ring failure bearing.

Inner ring failure

The same analysis can be done for the inner ring failure. Figure 4.4 shows that the inner ring failure is harder to identify, both with the autoencoder and the model-based approach. The error starts developing at around timestamp 1780. We can see that it is easier to detect the beginning of the error by looking at the autoencoder error than by looking at the characteristic frequency peak.



(A) (B)

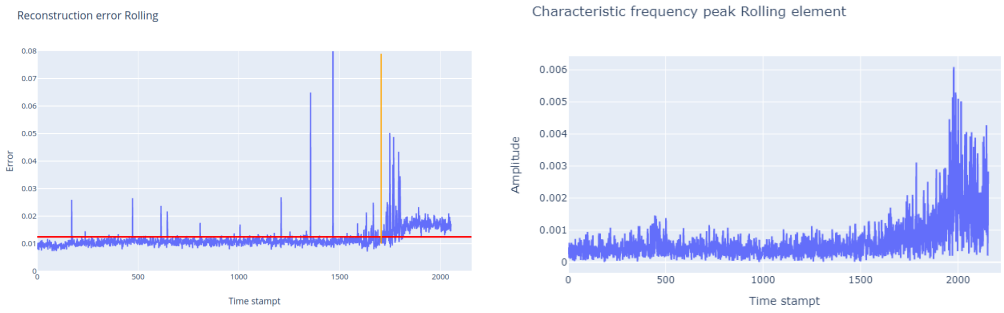
FIGURE 4.4: Evolution of the inner ring bearing failing frequency (A) and autoencoer error (B) for the inner ring failure bearing.

Rolling element failure

We repeat the analysis for the rolling element failure. Figure 4.5 shows that both methods predict that the error begins its development at around timestamp 700.

Healthy bearing

Finally, we look at the reconstruction error of a healthy bearing. In this case, there is no bearing failing frequency, since there is no error to look at. Figure 4.6 (A) shows that, except for some isolated outlier, the reconstruction error is below the threshold through all the experiment. Figure 4.6 (B) provides the timestamps when the failures begin its development. This information will be used to create the labels for the classification model.



(A) (B)

FIGURE 4.5: Evolution of the rolling element failing frequency (A) and autoencoder error (B) for the rolling element failure bearing.

Beginning of the failure		
Inner ring	Outer ring	Rolling el.
1780	760	1700

(A)

FIGURE 4.6: Evolution of the reconstruction error for the healthy bearing (A). Beginning of the failure development (B).

4.2 Siamese network

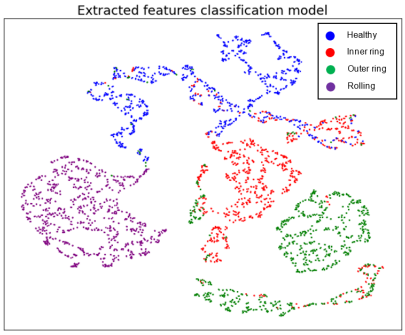
After obtaining the labels of the failure development, a bearing failure classification model is built. The first approach is to use a Siamese network. The training process is explained in Chapter 3. The results are assessed by looking at two aspects:

- **Visualization of the embeddings:** One goal of using a Siamese network is to create good embeddings for the waveforms. These embeddings should be close together for samples with the same failure and far apart for samples with different failure. One way to check this is by visualizing the embeddings in a two-dimensional plot. Since the embedding space is a 20-dimensional space, we use a dimensionality reduction technique to visualize the embeddings, called UMAP [42, 43]. In its simplest sense, the UMAP algorithm consists of two steps: construction of a graph in high dimensions followed by an optimization step to find the most similar graph in lower dimensions. For more information about the UMAP refer to the github of the project, or see [43].
- **Accuracy of the predictions:** The second measure of performance is the accuracy of the final predictions of the failure.

The experiments are done using the IMS data set (and the labels provided by the autoencoder) and also the Paderborn data set. Both data sets are mixed in order to simulate different operating conditions.

Pretraining

The first step of the training phase is to pretrain the algorithm using only the cross-entropy loss function. By doing so, we expect to have a good initialization of the embeddings.



(A)

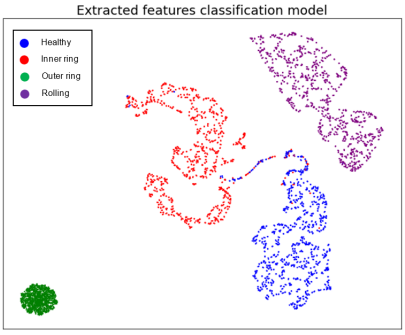
Acc.	Confusion matrix
0.90	$\begin{pmatrix} 539 & 67 & 144 & 0 \\ 28 & 960 & 0 & 2 \\ 25 & 17 & 661 & 2 \\ 0 & 0 & 0 & 590 \end{pmatrix}$

FIGURE 4.7: (A) Visualization of the extracted embeddings during the pretraining phase of the Siamese network. (B) Accuracy of the predictions.

Figure 4.7 shows that the embeddings are quite clustered by the failure mode. This means that the cross entropy loss already generates good embeddings. Also, the pretraining accuracy is also a good starting point.

Training

In this phase of the learning process, we aim to improve both the embedding representation and the accuracy of the model.



(A)

Acc.	Confusion matrix
0.97	$\begin{pmatrix} 716 & 34 & 0 & 0 \\ 25 & 965 & 0 & 0 \\ 0 & 0 & 705 & 0 \\ 0 & 3 & 0 & 587 \end{pmatrix}$

FIGURE 4.8: (A) Visualization of the extracted embeddings during the training phase of the Siamese network. (B) Accuracy of the predictions.

Figure 4.8 shows the results of training the Siamese network using contrastive loss for the embedding layers, and cross-entropy loss for the classifier layers. We observe that there is a clearer separation of the embedding points according to the failure modes. Moreover, the accuracy of the predictions has also increased to 0.97. Therefore, using the contrastive loss to design good embeddings has helped improve

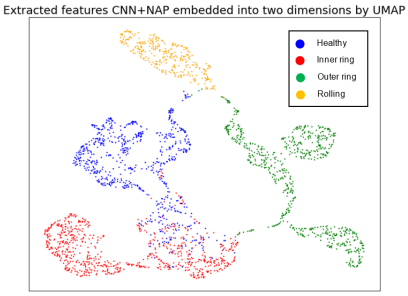
the classification process. Also, the model mostly confuses healthy and inner ring bearings. This is expectable because the inner ring failure is the most difficult to detect since it is the farthest to the sensors that record the vibration signal [2].

4.3 Triplet learning network

The second approach to build a classification model is to use a triplet learning strategies. Again, both IMS and Paderborn data sets are used to simulate multiple operating conditions. The learning process also contains a pretraining phase and a training phase. We will assess both the embeddings visualization and the classifier performance.

Pretraining

Figure 4.9 shows the results of the pretraining phase. Again, the pretraining the network with the cross-entropy loss produces a good initialization of the embeddings and a fairly good accuracy.

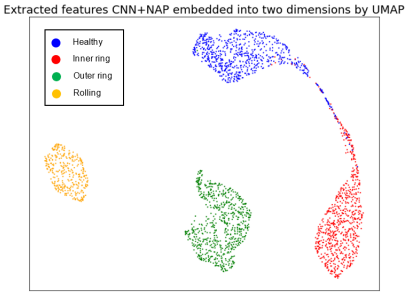


(A)

Acc.	Confusion matrix
0.90	$\begin{pmatrix} 627 & 45 & 2 & 76 \\ 35 & 667 & 1 & 37 \\ 25 & 5 & 653 & 22 \\ 0 & 0 & 2 & 338 \end{pmatrix}$

FIGURE 4.9: (A) Visualization of the extracted embeddings during the pretraining phase of the triplet learning network. (B) Accuracy of the predictions.

Training



(A)

Acc.	Confusion matrix
0.98	$\begin{pmatrix} 733 & 17 & 0 & 0 \\ 24 & 716 & 0 & 0 \\ 0 & 0 & 705 & 0 \\ 0 & 0 & 0 & 340 \end{pmatrix}$

FIGURE 4.10: (A) Visualization of the extracted embeddings during the training phase of the triplet learning network. (B) Accuracy of the predictions.

Figure 4.10 shows the results of the training phase. Triplet learning strategies allow to create very good embeddings, creating separated clusters according to the

bearing failure. Therefore, this feature extraction process helps the classifier to obtain better performance. The only confusion is again between the inner ring and healthy bearings. Moreover, the labels of the failure include the whole evolution of the bearing failure. Thus, some of the samples labelled as having a failure were at the beginning of the failure development. In fact, from the 24 miss classified samples, 18 were found between the timestamps 1780 and 1900 of the IMS data set, which corresponds to the early development of the failure. Therefore, triplet learning strategies improve the classification power of the network. We had implemented two triplet learning strategies: *batch all* and *batch hard*. Both strategies gave very similar results. The only difference is that batch hard strategy tends to collapse easier to one point in the embedding space, obtaining a loss value equal to the margin. Thus, having a good initialization of the embeddings and using low learning rates is very important to ensure convergence. On the other hand, when the network collapse is avoided, the network converges faster to zero by using batch hard strategy. However, for this project, it is very important to design a model which could be trained with few data samples. The batch all strategy is more suitable to accomplish this task since it creates more triplets from the same data set.

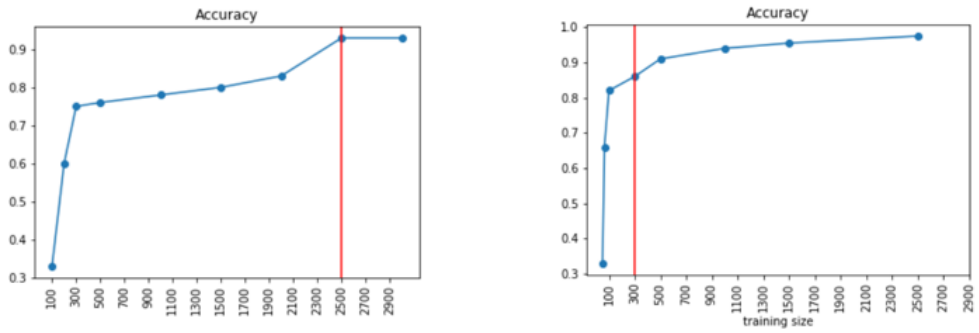
4.4 Comparison of classification models

We have seen in the previous sections that both the triplet learning models and the Siamese networks gave very similar performance results. The embeddings of the triplet learning network are more clearly clustered by the type of bearing failure. This suggests that the triplet learning strategy produces slightly better embeddings. However, the final accuracy of the triplet learning network was just 0.01 higher than the Siamese network accuracy. One possible explanation is that the data is easy enough to classify so that both strategies have the same performance. Therefore, we designed an experiment to compare both methods. The experiment consists in decreasing the training data by some proportion, training both models and comparing the results. We focused on the final accuracy of the predictions.

Figure 4.11 shows the change in the accuracy of the predictions when decreasing the sample size. We observe that the model trained using triplet learning is much more robust to smaller training sizes than the Siamese network. This happens because the batch all triplet strategy produces many more samples to feed the neural network than the training of the Siamese network. Therefore, triplet learning strategies are more suitable for small data sets.

4.5 Elastic weight consolidation

The last part of the project consisted of applying the elastic weight consolidation algorithm to continuously train the classification model. Usually, all the historical data is not available during training. Therefore, it is interesting to retrain the classification model when new data arrives. If the volume of data is small and can all be stored in a database, the best option is to retrain the model with the whole data set. However, when the amount of data is considerably big, training the model with all the historical data may be computationally inefficient, leading to too large training times. Moreover, the data may even be deleted from the database after a certain amount of time. Therefore, it is important to explore techniques that allow to retrain the model with new data without using all the data itself. Moreover, we want



(A) (B)

FIGURE 4.11: Accuracy of the classification models ((A) Siamese network and (B) triplet learning network) as a function of the training sample size.

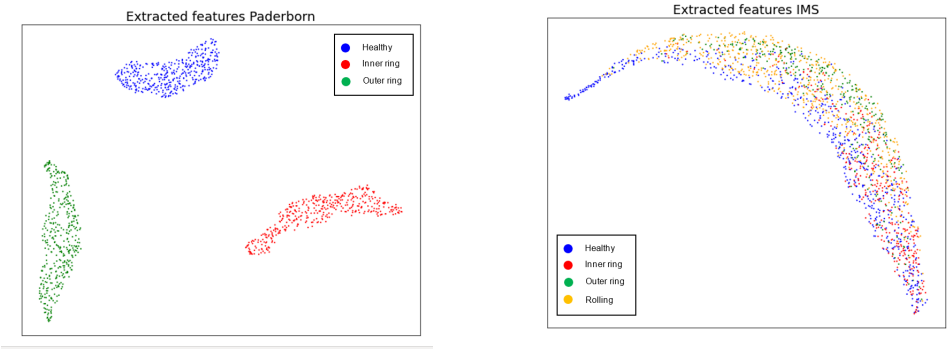
to model to learn new features of the data without forgetting about the previously learnt information. For these reasons we chose EWC. To test the performance of this algorithm, we perform the following experiment.

4.5.1 EWC-concept drift

In this experiment, we apply the EWC algorithm to two very different data sets. The first data set is used to train the initial model. Then, a second data set is used to retrain the model without forgetting about the first data set. Since the two data sets are very different, this experiment simulates concept drift. To simulate concept drift, we use two different data sets:

- **Paderborn data set.** This dataset will be considered as the *old* dataset.
- **IMS data set.** We will use this data as the *new* data, when the concept drift has already happened.

First, a classification model is trained using a triplet learning network, just as it was described in the previous section. After training the model with Paderborn data set, we check the performance of the model in both data sets. Figure 4.12 shows that

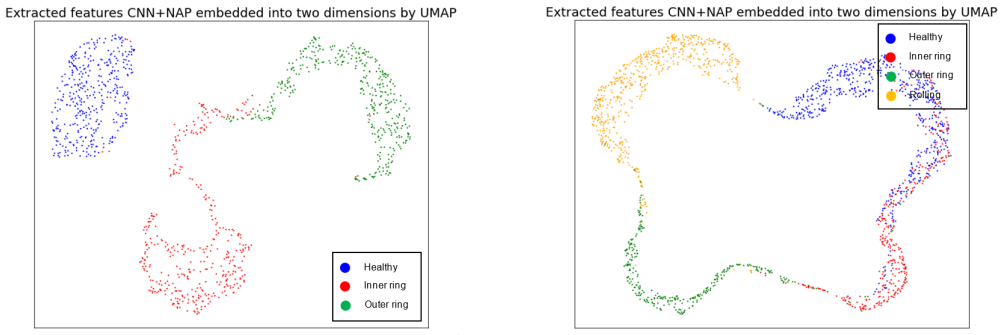


(A) (B)

FIGURE 4.12: Embeddings after training task A. (A) Paderborn data, (B) IMS data.

the embeddings are clearly clustered according to the bearing failure for the first

data set (Paderborn), while they are mostly random for the second data set (IMS). This happens because the two data sets are very distinct, and so training the network with one of them does not allow it to generalise well to the other. We want the model to learn to classify IMS failures as well. Moreover, table 4.1 (A) shows that the accuracy of the IMS data set is very poor. One option would be to sequentially train the model, first with Paderborn data set and then with IMS data set. However, if we train only 20 epochs with IMS data set, the accuracy of the model on the Paderborn data set falls to 0.40. Therefore, we need an algorithm which forbids the network from forgetting the first task. For this reason, we use the EWC algorithm. We begin by computing two empirical diagonalized fisher matrices: one for the embedding network, where the loss function is triplet loss, and one for the classifier layers, where the loss function is the cross-entropy loss. Then, we train the model with task B using the EWC penalty. First, we train the embeddings and then the classifier. The results are shown in Figure 4.13 and table 4.1 (B). Figure 4.13 shows that after training the model with IMS data, the embeddings of this data set improve a lot. Moreover, the embeddings of task A do not get much worse. Table 4.1 (B) shows that the model gives a similar accuracy for both data sets, even though it is not as good as it would be if we had trained the models with the two data sets at the same time. We conclude that this algorithm allows to learn the second task without forgetting too much about the first one. However, the results are better if the model is trained with both data sets.



(A) (B)

FIGURE 4.13: Embeddings after training task B. (A) Paderborn data, (B) IMS data.

After training task A	
	Accuracy
Task A (Paderborn)	1.0
Task B (IMS)	0.19

After training task B	
	Accuracy
Task A (Paderborn)	0.87
Task B (IMS)	0.90

TABLE 4.1: Performance of the model in tasks A (Paderborn) and task B (IMS).

Chapter 5

Conclusions

The monitoring of machine health is of great importance in the industry sector. Having an unexpected equipment failure can lead to very expensive and dangerous consequences. Examples of issues caused by machine failures are unplanned production downtime, costly replacement of machine components and even environmental problems. Rolling bearings are one of the most important components of most rotating machines and are one of the main causes of rotating equipment failures. For this reason, monitoring the health of rolling bearings has been a main research interest in the past few years.

Predictive maintenance techniques are used to detect potential defects in the earliest stages enabling the maintenance activity to be planned. In the case of bearing monitoring, vibration analysis can be very useful in diagnosing the bearing damage. This method is model-based, which means that it uses a physical model to predict the behaviour of the system. This method is simple and offers high performance. However, it requires some specific information about the bearings: the rotation velocity and the bearing failing coefficients. Sometimes the rotation velocity is unknown, usually because it varies depending on the operating condition. Moreover, the bearing failing coefficients, which should be specified by the manufacturer, may be unknown or may not correspond to the true coefficients. For these reasons, a new approach is needed to predict bearing failures when the characteristic information of the bearings is unknown.

In order to solve this challenge, we decided to use deep learning methods, since they form a data-based approach which does not need previous information about the system. Since the machines may be operating at different rotation velocities, we wanted our model to be robust to different operating conditions. Moreover, the model should be able to detect the early development of the failures. Finally, the model should be robust to small data sets. The training data consisted of a time sequence of vibration signals, called waveforms. At the beginning of the time series, all bearings were healthy, whilst in the end, some had suffered from a specific failure.

The first step was to build a model which allowed us to detect the beginning of a failure development. This model was then used to label the data sets so that the labels contained the complete evolution of the failure. The model consisted in a one-dimensional convolutional autoencoder. The task of the autoencoder was to create a reduced representation of the input and then reconstruct the input signal from its lower-dimensional representation. The autoencoder was trained only with healthy data so that it learnt to reconstruct *only* the healthy waveforms. When a new data sample was fed to the network, the reconstructed error was calculated. For healthy data samples, such error was small. On the other hand, the error started to increase

as the failure developed. By studying the probability distribution of the reconstruction error with healthy data, we defined a threshold, above which the data was considered to be unhealthy. We tested the performance of the model by comparing it to the output of the model-based approach. We saw that the autoencoder detected the beginning of the failure at the same time as the model-based, and sometimes even a bit earlier.

After obtaining the labels for the training data, we designed the core model of this project: a classification model for bearing failures. Such model had to be robust to different operating conditions. One way to accomplish this property was to divide the network into two parts: the embedding and the classifier. The embedding network would extract valuable features from the input in a reduced dimension space. This mapping should be made such that input samples suffering from the same failure were close together in the embedding space, and far from samples suffering from a different fault. These embeddings would help increase the performance of the classification model. Two different network designs were studied. The first approach consisted of a Siamese network, trained with contrastive loss. The network was fed with pairs of samples. If they had the same failure, they were pulled together in the embedding space. Otherwise, there were pulled apart. The second approach was to train the embedding network with triplet loss. This time, the network was trained with triplets of samples, two of them with the same label (positive) and the last one with a different label (negative). The goal of the embedding network was to create the embeddings such that a positive sample was closer to the other positive than to the negative sample. Triplet strategies allow to create more samples from the same training data since it creates triplets instead of pairs. Moreover, the triplets are created in an online manner, in such a way that only informative triplets are fed to the network. For these reasons, triplet learning usually gives better results. Both triplet and siamese strategies led to optimal performances in classifying the bearing failures. However, the triplet learning model was more robust to smaller training sizes, which made the model more suitable for solving our problem.

The last part of the project was to learn how to train the classification model continuously with time. Usually, all the training data is not available at the training stage. Therefore, it is useful to retrain the model as new data arrives. However, we do not want our model to forget the knowledge acquired from previous data. To train a model to learn a second task without forgetting about the first one, we applied an algorithm called Elastic Weight Consolidation (EWC). It provides a quadratic penalty, weighted so to give higher loss to the network parameters which were more important to train the first task. After applying this algorithm, the model had a similar performance for both tasks. This performance was a bit worse than the one obtained by training the model with the whole data set, but it was dramatically better than training the two tasks sequentially. Therefore, if both training sets are not available at the same time, EWC is a good alternative to learn both tasks.

In summary, we have used multiple deep learning methods to successfully predict the health status of rolling bearings. We believe that deep learning techniques can be a powerful substitute for vibration analysis when the characteristic information of the machines is missing.

Bibliography

[1] Bradley William Harris. "Anomaly detection in rolling element bearings via two-dimensional Symbolic Aggregate Approximation: Anomaly detection in rolling element bearings via two-dimensional Symbolic Aggregate Approximation". PhD thesis. Virginia Tech. URL: https://vtechworks.lib.vt.edu/handle/10919/23103.

[2] Alfonso Fernandez. *Vibration analysis learning*. 2017. URL: https://www.power-mi.com/content/rolling-element-bearings (visited on 04/04/2020).

[3] Robert B. Randall and Jérôme Antoni. "Rolling element bearing diagnostics—A tutorial". In: *Mechanical Systems and Signal Processing* 25.2 (2011), pp. 485–520. ISSN: 08883270. DOI: 10.1016/j.ymssp.2010.07.017. URL: http://www.sciencedirect.com/science/article/pii/S0888327010002530.

[4] Shen Zhang, Shibo Zhang, Bingnan Wang, and Thomas G. Habetler. "Machine Learning and Deep Learning Algorithms for Bearing Fault Diagnostics – A Comprehensive Review". In: *IEEE Access* 8 (2020). DOI: 10.1109/ACCESS.2020.2972859.

[5] Mark Gaudel. *Envelope analysis - the key to rolling-element bearing diagnosis (bo0187)*. 2001.

[6] A. Malhi and R. X. Gao. "PCA-based feature selection scheme for machine defect classification". In: *IEEE Transactions on Instrumentation and Measurement* 53.6 (2004), pp. 1517–1525.

[7] Z. Wang, Q. Zhang, J. Xiong, M. Xiao, G. Sun, and J. He. "Fault Diagnosis of a Rolling Bearing Using Wavelet Packet Denoising and Random Forests". In: *IEEE Sensors Journal* 17.17 (2017), pp. 5581–5588.

[8] David He, Li Ruoyu, Junda Zhu, and Mikhail Zade. "Data Mining Based Full Ceramic Bearing Fault Diagnostic System Using AE Sensors". In: *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council* 22 (Dec. 2011), pp. 2022–31. DOI: 10.1109/TNN.2011.2169087.

[9] A. Rojas and A. K. Nandi. "Detection and Classification of Rolling-Element Bearing Faults using Support Vector Machines". In: *2005 IEEE Workshop on Machine Learning for Signal Processing*. 2005, pp. 153–158.

[10] Sheng Guo, Tao Yang, Wei Gao, Chen Zhang, and Yanping Zhang. "An Intelligent Fault Diagnosis Method for Bearings with Variable Rotating Speed Based on Pythagorean Spatial Pyramid Pooling CNN". In: *Sensors (Basel, Switzerland)* 18.11 (2018). DOI: 10.3390/s18113857.

[11] Huijie Ma, Shunnming Li and Zenghui An. "A Fault Diagnosis Approach for Rolling Bearing Based on Convolutional Neural Network and Nuisance Attribute Projection under Various Speed Conditions". In: *Applied Sciences* 8:1603 (2019). DOI: 10.3390/app9081603.

[12] Karan Aggarwal, Onur Atan, Ahmed Farahat, Chi Zhang, Kosta Ristovski, and Chetan Gupta. *Two Birds with One Network: Unifying Failure Event Prediction and Time-to-failure Modeling*. arXiv: 1812.07142v1.

[13] M. Basseville and I. V. Nikiforov. *Detection of abrupt changes: Theory and application*. Prentice Hall information and system sciences series. Englewood Cliffs N.J.: Prentice Hall, 1993. ISBN: 0131267809.

[14] Wentao Mao, Jianliang He, Jiamei Tang, and Yuan Li. "Predicting remaining useful life of rolling bearings based on deep feature representation and long short-term memory neural network". In: *Advances in Mechanical Engineering* 10.12 (2018), p. 168781401881718. ISSN: 1687-8140. DOI: 10.1177/1687814018817184.

[15] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*. Vol. 1. 10. Springer series in statistics New York, 2001.

[16] Michael A. Nielsen. *Neural networks and deep learning*. Vol. 25. Determination press San Francisco, CA, USA: 2015.

[17] Ben Dickson. *What are artificial neural networks (ANN)?* 2019. URL: https://www.experfy.com/blog/what-is-artificial-neural-network-ann (visited on 04/05/2020).

[18] Facundo Bre, Juan M. Gimenez, and Víctor D. Fachinotti. "Prediction of wind pressure coefficients on building surfaces using artificial neural networks". In: *Energy and Buildings* 158 (2018), pp. 1429–1441. ISSN: 0378-7788. DOI: 10.1016/j.enbuild.2017.11.045. URL: http://www.sciencedirect.com/science/article/pii/S0378778817325501.

[19] Sebastian Ruder. *An overview of gradient descent optimization algorithms*. 2016. arXiv: 1609.04747 [cs.LG].

[20] Yann Lecun. "A theoretical framework for back-propagation". English (US). In: *Artificial neural networks*. Ed. by P. Mehra and B. Wah. IEEE Computer Society Press, 1992.

[21] Jordi Vitrià. *DeepLearningMaster Repository*. https://github.com/DataScienceUB/DeepLearningMaster20192020/blob/master/. 2019.

[22] Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. *Automatic differentiation in machine learning: a survey*. 2015. arXiv: 1502.05767 [cs.SC].

[23] Standfor University. *Convolutional Neural Networks (CNNs / ConvNets)*. 2020. URL: https://cs231n.github.io/convolutional-networks/ (visited on 04/05/2020).

[24] Will Badr. "Auto-Encoder: What Is It? And What Is It Used For? (Part 1)". In: *Towards Data Science* (22/4/2019). URL: https://towardsdatascience.com/auto-encoder-what-is-it-and-what-is-it-used-for-part-1-3e5c6f017726.

[25] Ahmed Fawzy Gad. "Image Compression Using Autoencoders in Keras | Paperspace Blog". In: *Paperspace Blog* (1/2/2020). URL: https://blog.paperspace.com/autoencoder-image-compression-keras/.

[26] Will Badr. "5 Ways to Detect Outliers That Every Data Scientist Should Know (Python Code)". In: *Towards Data Science* (5/3/2019). URL: https://towardsdatascience.com/5-ways-to-detect-outliers-that-every-data-scientist-should-know-python-code-70a54335a623.

[27] Zofia Hanusz, Joanna Tarasinska, and Wojciech Zielinski. "Shapiro-Wilk test with known mean". In: *REVSTAT - Statistical Journal* 14.1 (2016), pp. 89–100. ISSN: 16456726.

[28] Antonio trujillo ortiz and Rafael Hernandez-Walls. *D'Agostino-Pearson's K2 test for assessing normality of data using skewness and kurtosis.* Sept. 2003.

[29] S. Chopra, R. Hadsell, and Y. LeCun. "Learning a similarity metric discriminatively, with application to face verification". In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. 2005, 539–546 vol. 1. ISBN: 1063-6919. DOI: 10.1109/CVPR.2005.202.

[30] Keras team. *Keras examples*. https://github.com/keras-team/keras/tree/master/examples. 2019.

[31] Florian Schroff, Dmitry Kalenichenko, and James Philbin. "FaceNet: A Unified Embedding for Face Recognition and Clustering". In: (2015), pp. 815–823. DOI: 10.1109/CVPR.2015.7298682.

[32] Olivier Moindrot. *Triplet Loss and Online Triplet Mining in TensorFlow*. 2018. URL: https://omoindrot.github.io/triplet-loss (visited on 04/06/2020).

[33] Alexander Hermans, Lucas Beyer, and Bastian Leibe. *In Defense of the Triplet Loss for Person Re-Identification*. arXiv: 1703.07737.

[34] Alexey Tsymbal. "The problem of concept drift: definitions and related work". In: (2004). URL: https://www.semanticscholar.org/paper/The-problem-of-concept-drift\%3A-definitions-and-work-Tsymbal.

[35] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. "Overcoming catastrophic forgetting in neural networks". In: *Proceedings of the National Academy of Sciences of the United States of America* 114 13 (2017), pp. 3521–3526.

[36] Abhishek Aich. *Elastic Weight Consolidation: Nuts and Bolts*. URL: https://abhishekaich27.github.io/data/WriteUps/EWC_nuts_and_bolts.pdf.

[37] Ferenc Huszar. "Comment on "Overcoming catastrophic forgetting in NNs": Are multiple penalties needed?" In: *inFERENCe* (16/3/2017). URL: https://www.inference.vc/comment-on-overcoming-catastrophic-forgetting-in-nns-are-multiple-penalties-needed-2/.

[38] Guillaume P. Dehaene. *A deterministic and computable Bernstein-von Mises theorem*. 2019. arXiv: 1904.02505 [math.ST].

[39] Joseph L Doob. *Le calcul des probabilites et ses applications*. 1949, pp. 23–27.

[40] Center for intelligent maintenance systems. *PCoE Datasets*.

[41] Paderborn University. *Bearing Data Center*. 2016. URL: https://mb.uni-paderborn.de/en/kat/main-research/datacenter/bearing-datacenter/ (visited on 04/07/2020).

[42] Adam Pearce Andy Coenen. *Understanding UMAP*. URL: https://pair-code.github.io/understanding-umap/ (visited on 04/09/2020).

[43] Leland McInnes, John Healy, and James Melville. *UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction*. 2018. arXiv: 1802.03426.