INDUCTION MOTORS GREY BOX MODELLING WITH DEEP LEARNING

Submitted by:

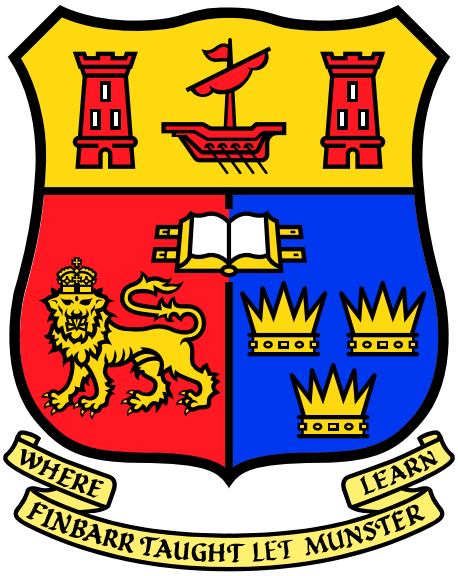
MARTIN RAMIREZ TINOCO

Supervisor:

DR ANDREA VISENTIN, PROF GREGORY PROVEN

Second Reader:

DR STEVEN PRESTWICH



MSc Data Science and Analytics

School of Computer Science & Information Technology
University College, Cork

September 2, 2022

Abstract

Deep learning techniques are improving in terms of reliability and effectiveness. Particularly, this dissertation studies two approaches in relation with an important and demanded application that focuses on rotating machinery fault diagnosis and classification. Implementing an AI system that can automatically detect faults at an early stage and recommend stopping of a machine to avoid unsafe conditions in the process and environment has become possible.

First Approach: for this scenario the problem consists in simulating precisely the data from an induction motor through Matlab using one of its tools called SimuLink. The resulting successful simulation will work hand in hand with a Deep learning model. SimuLink simulation will be capable of simulating different states of the induction motor, whereas the Deep learning model will be classifying these states appropriately. Moreover, for this approach, we used two induction motor databases: a database provided by UCC containing data of three different induction motors and MAFAULDA database.

Second Approach: this part only uses a well-known machinery fault database called MAFAULDA which is composed of 1951 multivariate time-series acquired by sensors on a SpectraQuest's Machinery Fault Simulator (MFS) Alignment-Balance-Vibration (ABVT). The 1951 comprises six different simulated states: normal function, imbalance fault, horizontal and vertical misalignment faults and, inner and outer bearing faults. Particularly, we focused on two accelerometers that measure the vibration of the induction motor (under-hang and overhang) and their three directions: tangential, axial and radial.

The main aim of this approach is to obtain a visualization of the important parts in the frequency domain of multi-variate time series that trigger the decision in our Deep learning model. With this insight, the possible technician will be able to identify and foresee if the motor is functioning at an unhealthy behavior and stop it if this happened.

In more detail, firstly we apply Fast Fourier Transformation in order to change our input from the time domain to the frequency domain and also, as a dimensionality reduction technique. Secondly, we built, trained and tuned a famous Deep learning technique called 1-D Convolutional Neural Network that classifies different states of the motor. Finally, we use Grad-CAM on the frequency domain of each class in order to highlight the important parts of that trigger the decision of the CNN.

Key words: Fault Detection, Artificial Intelligence, Grad-CAM, CNN, SimuLink, MatLab and Fast Fourier Transformation.

Declaration

I confirm that, except where indicated through the proper use of citations and references, this is my original work and that I have not submitted it for any other course or degree.

Signed: Martin Ramirez Tinoco

Martin Ramirez Tinoco
September 2, 2022

Acknowledgements

I would like to express my deepest appreciation to Dr. Andrea Visentin and Prof. Gregory Proven for having helped, guided and given feedback during this whole thesis. Additionally, this endeavor would not have been possible without the generous support from University College of Cork, who financed my research thesis.

I am also appreciative of the editing assistance, late-night feedback sessions, and emotional support I received from my classmates and cohort members, especially my class mates. Also deserving of recognition are, the university's librarians, research assistants, and study participants who had an influence and motivated me.

Lastly, I would be remiss in not mentioning my family, especially my girlfriend, parents and brother. Their belief in me has kept my spirits and motivation high during this process. I would also like to thank my cat for all the entertainment and emotional support.

Dedication

This dissertation work is specially dedicated to Dr. Andrea Visentin. He has been an undeniably source of support and feedbacks through not only my thesis but also my Master Degree. Moreover, it should be said that he is one of the best professor I have had during my entire education. His attention to detail, experience and dedication thrived me to like even more Computer Science techniques such as those described in this paper. Finally, I would like to highlight also his dedication when it comes to help me, since, every time that I needed something from UCC, it was a matter of time till Andrea put these at my disposal for continuing and progressing my thesis.

Contents

List of Figures

Chapter 1

Introduction

Rotating machinery now plays a crucial role in many aspects of modern life, including manufacturing, transportation, and construction. In order to avoid premature failure, which can lead to major financial losses and safety issues, reliability in these equipment is crucial. Consequently, a successful monitoring strategy and accurate problem identification are essential to ensuring the rotating machines' long-term healthy operation. As a result, there has been a lot of study done on fault detection in rotating machinery in order to create a highly accurate real-time automatic recognition system.

An intelligent system and the detection of a vibration signal can determine the state of the machine and foretell potential failure. Therefore , it is feasible to create a system that will aid the maintenance team in correctly predicting and promptly fixing rotating equipment. Unplanned machine maintenance can cost money, take a lot of time, and require materials. To prevent a sudden system failure or breakdown, it is essential to foresee and identify machine failure in good time.

Thus, machinery has been continuously improving, and the adoption of new technologies has made it possible to monitor their performance. The ability to accurately and at an early stage determine whether a machine is operating in unhealthy conditions has been made possible by reductions in the cost of sensors and data storage, as well as increases in computing power and new developments of Deep learning algorithms.

Particularly, internal, external, or environmental factors can all be major causes of problems in induction motors (IMs). For instance, internal problems are typically categorized based on where they originate, such as electrical and mechanical, or where they manifest, such as stator and rotor. Figure 1.1 shows a fault tree where faults in IMs are categorised according to their origin (mechanical or electrical) [Benbouzid 2000]. Some of these failures, as mentioned, can cause down times and huge economical loses for the company involved. Figure 1.2 ilustrates the most common reasons that trigger a breakdown of Induction Motors. As can be seen from the picture, 51% of times, they are bearing faults related. Them, are followed by those related to stator winding and external conditions damages, which each account for 16% of faults. Rotor-related failures represent 5% and other damages 12% [Terron-Santiago et al. 2021]

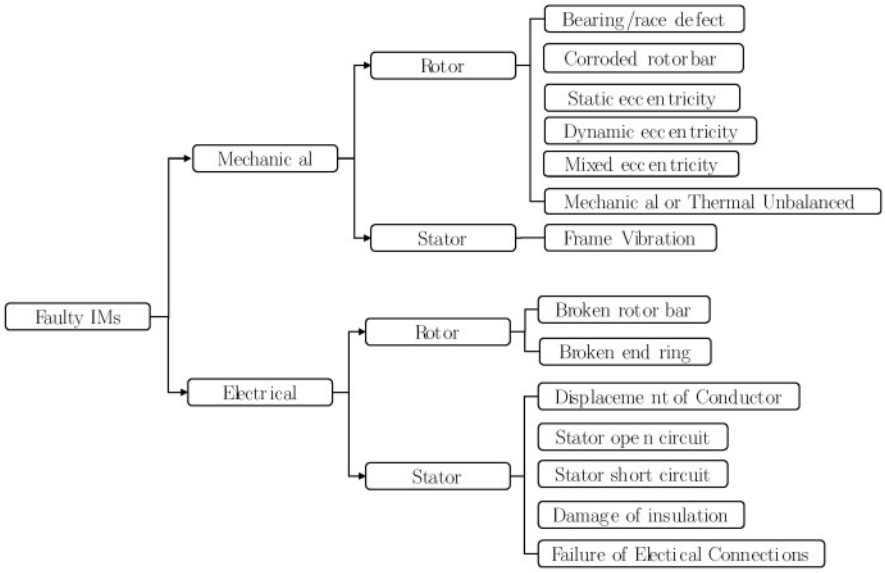


Figure 1.1: Summary of different failures in Induction Motors

Firstly, in this thesis, we simulated the behavior of an induction motor using SimuLink. We were able to set it up in such a manner that the input of our simulation was the frequency at which the motor spun (Hz). In addition, since the data from UCC recorded the stator magnetic flux, vibration, and current, we built the simulation having these parameters as outputs signals that then, will be compared to see how precisely our simulation worked. Moreover, should be highlighted that noise was present. Thus, when it comes to removing the noise and smooth our UCC data, we applied Fast Fourier Transformation along with Band Pass. Using Band Pass allowed us to highlight the frequencies that were contributing the most to each signal.

The results obtained for this database were good and precise. So, to double-check if our simulation was appropriate we worked out the same scenario with MAFAULDA database. Here, it was when the issues arose and we had to change the scope of the thesis. For instance, MAFAULDA database measures the vibration of the induction motor using two accelerometers which we were not able to simulate using SimuLink. Therefore, in order to have a comparison for the data we applied Fast Fourier Transformation and a regression of Sine function using Curve Fitting. With this study, it was found that Fast Fourier Transformation and Band Pass were not acceptable since we were constraining and losing much information from the data (around 60% of all the information).

Secondly, it was only studied MAFAULDA database [Ribeiro 2022]. The Machinery Fault Database (MAFAULDA) was selected to evaluate the performance of the proposed prediction models. This database imitates problematic failure scenarios. Moreover,

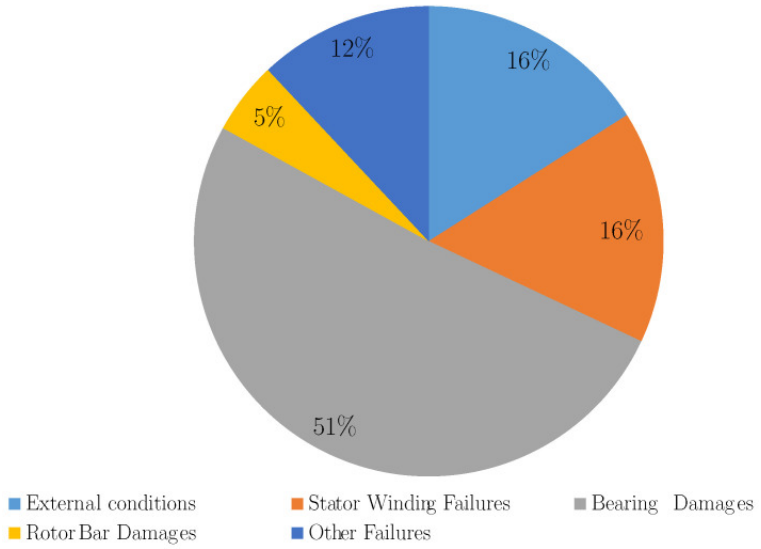


Figure 1.2: Most common reasons for down times on Induction Motors

this database is composed of 1951 multivariate time-series acquired by sensors on a SpectraQuest's Machinery Fault Simulator (MFS) Alignment-Balance-Vibration (ABVT). The 1951 comprises six different simulated states: normal function, imbalance fault, horizontal and vertical misalignment faults and, inner and outer bearing faults. As follows, in Figure 1.3 we can see the machine in charge of simulation an IM behaviour [Alzghoul et al. 2021].

This study aims to help at an early stage a possible technician to assess if the functioning of the motor is healthy or not. This will be achieved with explanatory heatmaps that highlight different frequency domain zones for each state of an induction motor described in MAFAULDA. These heatmaps would allow the technician to allocate healthy frequency zones and if the motor did not spin at these, that would mean that the motor would be functioning in an unhealthy state. Moreover, since it studied different failures, the technician would be able not only to identify unhealthy functioning but also, to classify this unhealthy state into one of the 5 different types of failures described at MAFAULDA.

In particular, 1-D CNN was trained and used to identify and classify the cause of faults in an induction motor, for different rotational speed, load level, and securities. To achieve this goal, the CNN model was developed and applied, in which different adjustments were made to the convolution and pooling layers aiming to fine-tune the hyperparameters.

In addition, this approach presents an explainability tool (Grad-CAM) in order to gain an outstanding visualization of the frequency domain spectrum of the vibration

signal. With this visualization we can perform an average heatmap for each type of state in MAFAULDA database and we can determine which parts of the frequency spectrum trigger a single data point to be classified into a class. Additionally, this tool is useful since it allows the possible technician to detect at an early stage whether or not the induction motor is functioning in a healthy zone of the vibration signal's spectrum.

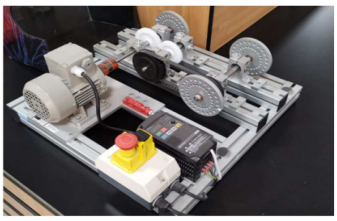


Figure 1.3: Machine Fault Simulator

The contents of this thesis are as follows: Chapter 2 describes the literature review, Chapter 3 presents the background required to fully understand this thesis, Chapter 4 depicts how we simulated the behavior of an induction motor in SimuLink and what the issues that made changing our scope were, Chapter 5 presents the pre-processing technique used (Fast Fourier Transformation) along with the Deep learning Model, Chapter 6 describes the experimental results obtained from MAFAULDA dataset and Chapter 7 describes our conclusion with some possible insights that might lead further investigations.

1.1 Environment Details

All these Deep learning models were built and executed using Colab's Pro GPU and RAM, having a good storage to handle the massive amount of data. Also, when it comes to the data itself, it was stored in a Drive and then, mounted into Colab. Moreover, we had to handle with dynamic memory in order to not run out of memory, for instance, we had to delete Pandas DataFrames as soon as they were not needed anymore.

Chapter 2

Literature Review

Fist of all, I would like to highlight some of the papers that I consider fascinating treating the topic of simulating induction motor using SimuLink.

[Shi et al. 1999] This paper describes a generalized model of the three-phase induction motor and its computer simulation using MATLAB/SIMULINK. Constructional details of various sub-models for the induction motor are given and their implementation in SIMULINK is outlined. Direct-on- line starting of a 7.5-kW induction motor is studied using the simulation model developed. From this paper, we could see the difficulty of simulating through SimuLink the mathematics that an induction motor carries under the hood.

Moreover, referring to [Phukon, and Baruah 2015], where we took inspiration from when it come to trying implementing this complicated bunch of mathematical formulas that an induction motor. In particular this paper describes induction motors' efficient design and other benefits like dependable operation, low initial costs, ease of use and simple maintenance, high efficiency, and the availability of simple and direct control gear for starting and speed control, induction motors are the most frequently used motor drives in industry. The usefulness of this motor has led to much research into the machine's transitory behavior. This article uses dq0 axis transformations of the stator and rotor variables in the arbitrary reference frame to demonstrate an induction machine implementation in Simulink step-by-step. First, the key formulas are presented, and then a generalized model of a three-phase induction motor is created and put into practice using these formulas.

As previously mentioned in this thesis, as reliability and accuracy of Deep learning techniques like CNN, Generate Adversal Neural Networks, and predictive maintenance have increased, fault detection in Induction Motors has become an obviously interesting topic. The following is a description of some of the most fascinating publications I came across while researching the subject.

In [Zhang, Peng, et al. 2017], a method called Deep CNN with First-Layer Kernels was

used to treat the input data, which were raw vibration signals, and used wide kernels in the first convolutional layer to characterise the input signal and suppress high-frequency noise.

Neural Networks with Training Interference (TICNN) is a technique that was created by the authors of [Zhang, C. Li, et al. 2018] to be used in signal-noisy situations, such as those seen in industrial settings. The White Gaussian function was used to amplify the signal with noise. Six interpolated convolution and pooling layers, one fully linked layer, and one softmax layer were employed to tackle the issue. In order to achieve the goal of noise filtering, as described in the study, it configured the kernel with 64x1 dimensions in the first layer and 3x1 dimensions in the subsequent layers. The CWRU database was used to evaluate the approach, and the results showed an accuracy of 96.1%.

Reference [Janssens et al. 2016] presented a study that involved the creation of a method that, with no assistance from a person, can figure out the characteristics of raw data that represent a machine failing. Various types of bearing failure, including failure in external raceways, lubrication degradation, healthy-bearing failure, and rotor imbalance, were examined during the experiment. Several experiments on various bearing types and failures were carried out to confirm the generality of the model. Two systems were compared utilising the same data in the study's conclusions. The precision values achieved by the author's system, which combined CNN and feature learning, were 93.61% and 87.25%, respectively, and were significantly higher than those of another system, which combined feature learning and Random Forest classifier.

Reference to [Jia, Lei, J. Lin, et al. 2016] proposed a Deep Neural Network (DNN) based method for both, fault characteristic mining and intelligent fault diagnosis of rotating machines. Viana et al. [Pestana-Viana et al. 2016] used kurtosis and entropy measures in their classification model across different fault situations and tested their influence on fault diagnosis. [Zhang, X. Li, et al. 2020] developed a DNN-based technique that controls imbalance classes using Synthetic Minority Over-sampling Technique (SMOTE) and Generative Adversarial Networks (GAN). The GAN generates additional fake samples to balance and further expand the training dataset, and accordingly improve overall classifying accuracy.

The most recent research in CNN visualisations, model trust evaluation, and weakly-supervised localisation is used in our work. CNN visualisation. Previous efforts [Simonyan, Vedaldi, and Zisserman 2014], [Springenberg et al. 2014] , [Zeiler, and Fergus 2014], [Gan et al. 2015] have highlighted "critical" pixels in CNN predictions to show where changes in their intensities have the most effects on the prediction's score. Specifically, Guided Backpropagation [Simonyan, Vedaldi, and Zisserman 2014] and Deconvolution [Springenberg et al. 2014] modify 'raw' gradients to provide qualitatively better results, whereas Simonyan et al. [Zeiler, and Fergus 2014] show partial derivatives of projected class scores with respect to pixel intensities. These approaches are compared in [Mahendran, and Vedaldi 2016]. Despite producing fine-grained visualizations, these methods are not class-discriminative.

Weakly supervised localization in the context of CNNs is a related area of research where the aim is to locate objects in pictures using just full image class labels [Cinbis, Verbeek, and Schmid 2016], [Oquab, Leon Bottou, et al. 2014], [Oquab, Léon Bottou, et al. 2015] and [Zhou et al. 2016].

The Class Activation Mapping (CAM) technique to localization is the one that most closely relates to our strategy [Zhou et al. 2016]. Convolutional layers and global average pooling [M. Lin, Chen, and Yan 2013] are used in this method to modify CNN architectures for image classification, which results in class-specific feature maps. Other researchers have looked at related techniques employing global max pooling and log-sum-exp pooling [Oquab, Léon Bottou, et al. 2015] and [Pinheiro, and Collobert 2015].

The limitation of CAM is that it only applies to a specific type of CNN architectures that perform global average pooling over convolutional maps right before prediction (i.e. conv feature → maps global average pooling → softmax layer). This is because CAM requires feature maps to come before softmax layers directly. On some tasks (like image classification), such structures may perform less accurately than generic networks, or they may simply not be relevant to any other tasks (e.g. image captioning or VQA). We provide a novel gradient-based feature map fusion method that does not involve changing the network architecture. Due to this, our method may be used with any CNN-based architecture, such as those used for picture captioning and visual question answering. Grad-CAM reduces to CAM for an architecture with complete convolution. Grad-CAM is a generalisation of CAM as a result.

Chapter 3

Background

3.1 Fast Fourier Transformation

This technique was used in this paper as a dimensionality reduction technique, allowing us to handle and process the raw data in a more efficient and quicker way. In addition, since we aim to have an explainability in our problem, applying FFT allows us to convert the input of multi-variate time series from the time domain to the frequency spectrum. For further explanation [Heckbert 1995].

3.1.1 Definition of Fourier Transformation

The Fourier Transform (FT) of the function $f(x)$ is the function $F(w)$, where:

$$F(w) = \int_{-\infty}^{\infty} f(x) e^{-iwx} dx \tag{3.1}$$

and the inverse Fourier Transoform is:

$$f(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} f(w) e^{iwx} dw \tag{3.2}$$

Recall that $i = \sqrt{-1}$ and $e^{i\theta} = cos\theta + sin\theta$

Think of it as a transformation into a different set of basis functions. The Fourier transform uses complex exponentials (sinusoids) of various frequencies as its basis functions. (Other transforms, such as Z, Laplace, Cosine, Wavelet, and Hartley, use different basis functions).

A Fourier transform pair is usually written $f(x) \leftrightarrow F(w)$, or $F(f(x)) = F(w)$ where F is the Fourier transform operator.

If $f(x)$ is though of as a signal (i.e. input data) the we call $F(w)$ the **signal's spectrum**.

3.1.2 Discrete Fourier Transform (DFT)

When a signal is discrete and periodic, we do not need the continuous Fourier transform. This is our case, since the signals recorded by the two acceleremoters located on the simulator machine were obtained at 50KHz which means we have 250000 data points (i.e. discrete). Instead we use the discrete Fourier transform, or DFT. Suppose our signal is a_n for n = 0 , ..., N - 1, and $a_n = a_{n+jN}$ for all n and j. The Discrete Fourier Transform of a, also known as the spectrum of a, is:

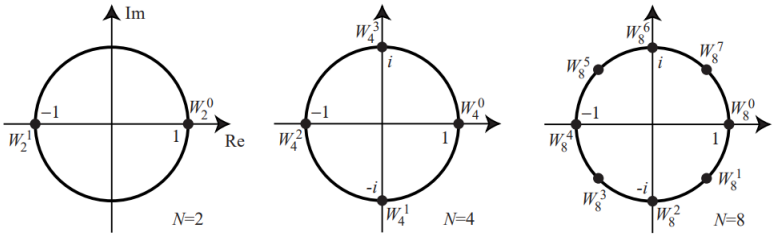
$$A_k = \sum_{n=0}^{N-1} e^{-i\frac{2\pi}{N}} a_n \tag{3.3}$$

This is more commonly written as:

$$A_k = \sum_{n=0}^{N-1} W_N^{kn} a_n \tag{3.4}$$

where $W_N^{kn} = e^{-i\frac{2\pi}{N}}$

and W_N^{kn} for k = 0, ..., N-1 are called *Nth roots of unity*. They are called this because, in complex arithmetic, $(W_N^k)^N = 1$ for all k. Theyâre vertices of a regular polygon inscribed in the unit circle of the complex plane, with one vertex at (1, 0). Below are roots of unity for N = 2, N = 4, and N = 8, graphed in the complex plane.



Powers of roots of unity are periodic with period N, since the Nth roots of unity are points on the complex unit circle every $2\pi/N$ radians apart, and multiplying by W_n is equivalent to rotation clockwise by this angle. Multiplication by W_N^N is rotation by 2π radians, that is, no rotation at all. In general, $W_N^k = W_N^{k+jN}$ for all integer j. Thus, when raising W_N to a power, the exponent can be taken modulo N.

The sequence A_k is the discrete Fourier transform of the sequence a_n. Each is a sequence of N complex numbers.

The sequence an is the inverse discrete Fourier transform of the sequence A_k. The formula for the inverse DFT is

$$a_n = \frac{1}{N} \sum_{k=0}^{N-1} W_n^{-kn} A_k \tag{3.5}$$

The formula is identical except that a and A have exchanged roles, as have k and n. Also, the exponent of W is negated, and there is a 1/N normalization in front.

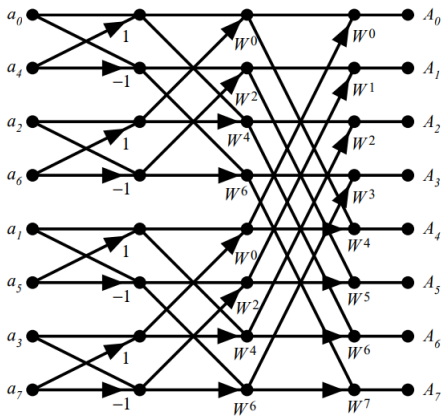
3.1.3 The Fast Fourier Transform (FFT) Algorithm

The FFT is a fast algorithm for computing the DFT. If we take the 2-point DFT and 4-point DFT and generalize them to 8-point, 16-point, ..., 2^r point, we get the FFT algorithm

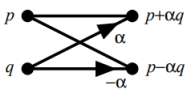
To compute the DFT of an N-point sequence using equation 3.4 would take $O(n^2)$ multiplies and adds. The FFT algorithm computes the DFT using $O(N \log N)$ multiplies and adds. Thus, we used FFT because of this reduction in the number of operations, making the problem to execute faster without losing much information.

There are many variants of the FFT algorithm. We will discuss one of them, the âdecimation in-timeâ FFT algorithm for sequences whose length is a power of two (N = 2^r for some integer r).

Below is a diagram of an 8-point FFT, where $W = W_8 = e^{-i\pi/4} = (1-i)/\sqrt{2}$



Butterflies and Bit-Reversal. The FFT algorithm decomposes the DFT into $\log_2 N$ stages, each of which consists of *N/2 butterfly computations*. Each butterfly takes two complex numbers p and q and computes from them two other numbers, $p + \alpha q$ and $p - \alpha q$, where α is a complex number. Bellow is a diagram of a butterfly operation.



In the diagram of the 8-point FFT above, note that the inputs are not in normal order: $a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7$, theyâre in the bizarre order: $a_0, a_4, a_2, a_6, a_1, a_5, a_3, a_7$

Below is a table of j and the index of the jth input sample, n_j:

j	0	1	2	3	4	5	6	7
n_j	0	4	2	6	1	5	3	7
j base 2	000	001	010	011	100	101	110	111
n_j base 2	000	100	010	110	001	101	011	111

The pattern is obvious if j and n j are written in binary (last two rows of the table). Observe that each n j is the bit-reversal of j. The sequence is also related to breadth-first traversal of a binary tree. It turns out that this FFT algorithm is simplest if the input array is rearranged to be in bit-reversed order. The re-ordering can be done in one pass through the array a:

```
for j = 0 to N-1
    nj = bit_reverse(j)
    if (j<nj) swap a[j] and a[nj]
```

General FFT and IFFT Algorithm for $N = 2^r$. The previously diagrammed algorithm for the 8-point FFT is easily generalized to any power of two. The input array is bit-reversed, and the butterfly coefficients can be seen to have exponents in arithmetic sequence modulo N. For example, for N = 8, the butterfly coefficients on the last stage in the diagram are $W_0, W_1, W_2, W_3, W_4, W_5, W_6, W_7$. That is, powers of W in sequence. The coefficients in the previous stage have exponents 0,2,4,6,0,2,4,6, which is equivalent to the sequence 0,2,4,6,8,10,12,14 modulo 8. And the exponents in the first stage are 1,-1,1,-1,1,-1,1,-1, which is equivalent to W raised to the powers 0,4,0,4,0,4,0,4, and this is equivalent to the exponent sequence 0,4,8,12,16,20,24,28 when taken modulo 8. The width of the butterflies (the height of the âXâsâ in the diagram) can be seen to be 1, 2, 4, ... in successive stages, and the butterflies are seen to be isolated in the first stage (groups of 1), then clustered into overlapping groups of 2 in the second stage, groups of 4 in the 3rd stage, etc. The generalization to other powers of two should be evident from the diagrams for N = 4 and N = 8.

The inverse FFT (IFFT) is identical to the FFT, except one exchanges the roles of a and A, the signs of all the exponents of W are negated, and thereâs a division by N at the end.

3.2 Convolutional Neural Network (CNN)

This section will present the details of the development, building, training, validation and test tasks of the proposed CNN model, as well as the theory and background of the main mathematics, computational principles and techniques used in this study. This section will begin by explaining the general theory and its constituent parts of Deep learning and CNN. Each of the modules that make up the developed model will be described in detail below. Afterwards, it will describe the methodology of the CNN model. It will be described for 2-D CNN but, the ideas and definitions for 1-D CNN are pretty much the same.

For further information refer to [Souza et al. 2021]

3.2.1 General theory description

Deep learning models are a subset of the machine-learning algorithm class, which is based on the cascade of multiple layers used in Artificial Neural Network (ANN) for learning in a supervised and/or unsupervised manner. In particular, 1-D CNN is fully capable of extracting features and properties that signal data has hidden under the hood. So that, it is a good choice to consider implementing 1-D CNN for our problem since we are dealing with multi-variate time series (signals).

CNN specially uses tensor operations (multiplication of matrices) to do forward propagation, whereas, it uses partial derivatives in order to update its weights (backward propagation). At a high level, a CNN structure usually consists in combinations of convolutional layers, nonlinear activation, and downsampling layers (such as MaxPooling) between the input and fully connected layer, as shown in Figure 3.1.

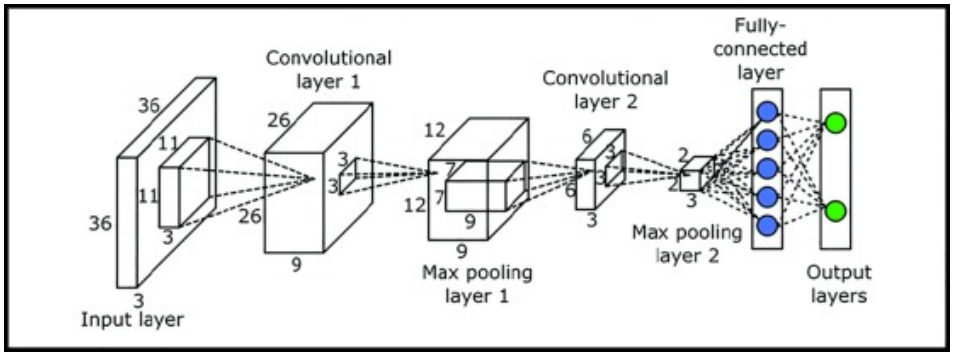


Figure 3.1: Schematic representation of the operations performed in CNN.

CNN is widely used in the field of computer vision, which features spatially shared weights and spatial subsampling (pooling) with two-dimensional (2D) or three-dimensional

sequence. Further, it has also been used in speech, audio, text, and time-series applications with **one-dimensional (1D)** sequence. In contrast to images, which are spatially distributed in two dimensions, the sensor output signals used in the domain of electrical-engineering applications are more frequent and often found to be 1D time-series data.

In addition to its strong performance, CNN tends to be computationally efficient because it requires fewer parameters than other architecture. Further, it is easier to parallelise across graphics processing unit cores. In addition, it requires a smaller pre-processing time than the multi-layer method because of the sharing of weights by employing a much smaller convolution kernel, which acts on a local region of the input data and extracts meaningful feature descriptors. When a single-layer CNN is used, the characteristics of the input signals are extracted through a signal convolution operation using a filter (or kernel). Detecting patterns captured by the kernel is possible regardless of where the pattern occurs using a convolutional operation. In CNNs, kernels are optimised to find the best parameters and to analyse the result as part of the supervised training process (we will do that using Talos, a hyperparametrizing technique). The map of the characteristics consists of an array of units (or layers) whose units share the same parameters (weight and bias vector), their activation generates the result of the kernel convolution using all the input data.

The name CNN indicates that it employs a mathematical operation called convolution that is a specialized type of linear operation. Therefore, CNN are neural networks that use convolution instead of general matrix multiplication in at least one of its layers. The operation used in CNN does not exactly match the definition of convolution used in other fields, such as pure engineering or mathematics.

The convolutional function is defined as

$$s(t) = \int_{a=-\infty}^{\infty} x(a)\, w(t-a)\, da \tag{3.6}$$

where w is the weight (kernel), x is the input value, t is the time, and the function $s(t)$ is referenced as feature map.

As the convolution function is cumulative and can be used on more than one cartesian axis at a time, as is the case with 2D image processing. The Eq. (2) describes equivalence and best represents the convolution function in machine learning applications [Goodfellow, Bengio, and Courville 2016].

$$S_{(i,j)} = (x * w)_{(i,j)} = \sum_{m} \sum_{n} x(i-m, j-n)\, w(m,n) \tag{3.7}$$

Convolutional layers consist of multiple local filters using raw input data and generate the local features of these data. The subsequent polling layers extract resources with a fixed length over sliding-data windows following various rules such as the mean, maximum, and so on [Ordóñez, and Roggen 2016].

Depending on the application, a number of combinations of convolution, nonlinear activation function, and subsampling layers are cascaded to obtain deeper network architecture. Feature descriptors extracted from the final pooling layers are used as input to the fully connected layer of CNN to obtain the probabilistic output using the

softmax function (that is the one that we will use since our problem is to classify different states of the Induction Motor)

A typical layer of a convolutional network consists of phases. In the first phase, the layer performs the convolution process in parallel to produce a set of linear activations. In the second phase, each linear activation is performed using a non-linear activation function such as ReLU (this is the one we are going to use in our model since our input was Scaled using Min Max Scaler). In the third phase, a pooling function is used to modify the output to the next or last layer. On the last phase, the activation function softmax is performed on the fully connected layer that classify the data.

The ReLU activation function is available in the convolution layer and is responsible for finding a non-linear function that represents the input signal (x), normalizing the weight (w) and accelerating the model convergence process.

The ReLU activation function has been more widely used than sigmoidal functions (in this case also, since we don't have negative values and so, it is not the most appropiate). The sigmoidal functions compress the output to a short interval and the ReLU function cancels all negative values, being linear to the positive ones, as shown in Figure 3.2.

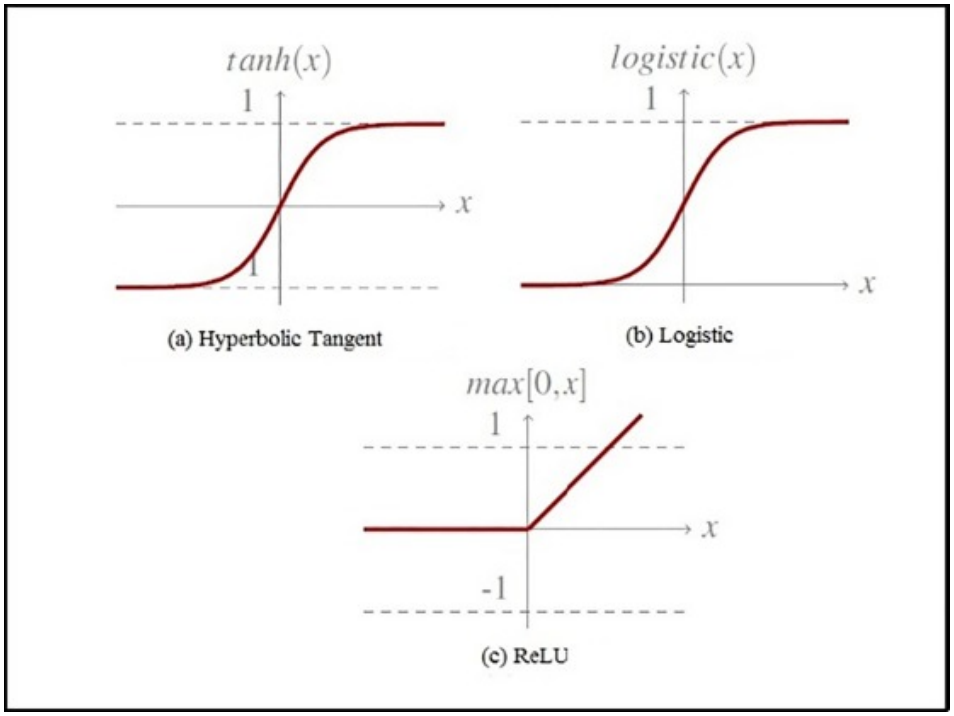


Figure 3.2: Different Activation Functions.

ReLU is the AF that will be used in our experiment it is worth to define it mathematically. This is as follows:

$$ReLU = max(x, 0) \text{ for } x \in \mathbb{R} \tag{3.8}$$

On the pooling layer, the most commonly function used is the maxpooling, with the aim of reducing the amount of CNN parameters, reducing the size of the data resulting from the convolution layer and accelerating the computing process. The sliding step size is configured in the function to scroll the entire length of the data volume [Jia, Lei, Lu, et al. 2018].

The most common form of the pooling layer is to replace the values of a region by its function (MaxPooling, GlobalAvgPooling, GlobalMaxPooling). Figure 3.3 shows a reduction of the data input volume from 4 x 4 to 2 x 2, due to the use of the 2 slide step, using the MaxPooling function.

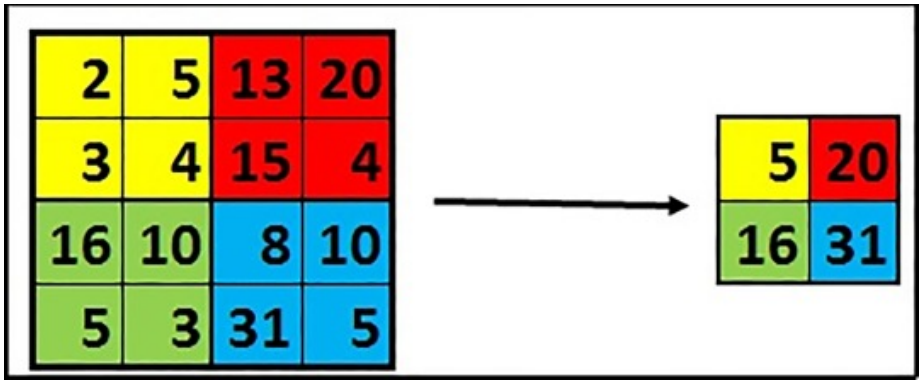


Figure 3.3: Example of MaxPooling function with dimension equal 2.

The fully connected layer is configured after the convolution and pooling layers, being the last layer of the network. Output data from the pooling layer is routed to this layer in order to sort the data and propagate the signal. These layers are similar to a conventional artificial neural network that uses softmax activation functions on the last output layer. Figure 3.4 shows the fully connected layer and softmax activation functions. Should be said also, that the number of neurons as output is equal to the number of classes that we want to classify in the problem.

The softmax function calculates the probability distribution of the event over 'n' different events, as Figure 3.5. The main advantage of using softmax is the output probability range of 0 to 1 and the sum of all probabilities equals one.

Dropout function is implemented during the training stage. To avoid overfitting and to eliminate the co-dependence between neurons that can be developed during the

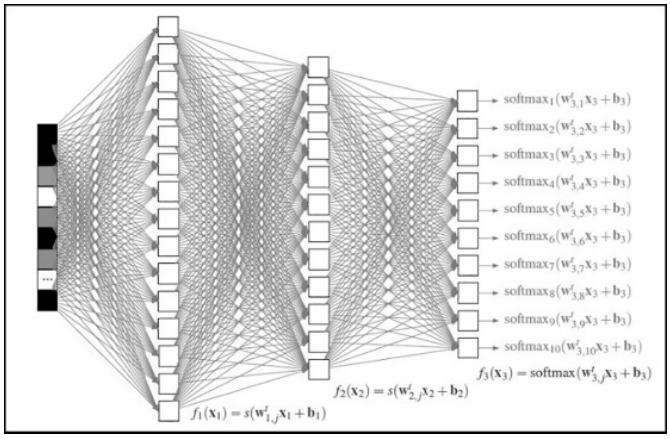


Figure 3.4: Fully connected network with Softmax Function

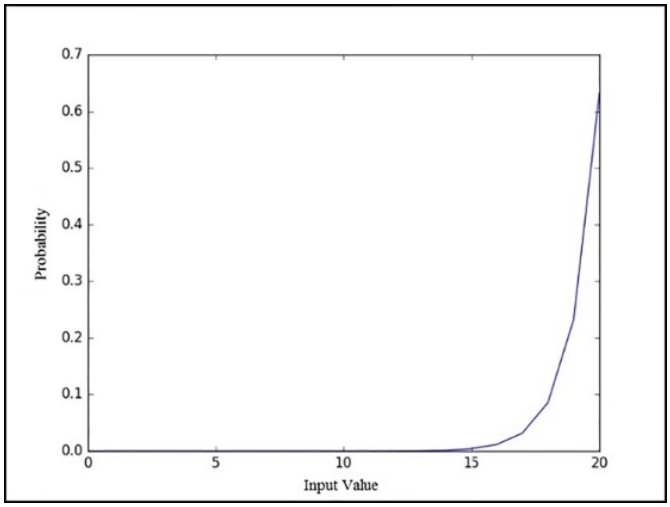


Figure 3.5: Softmax Function

learning process, it disables part of the neurons in a layer so that they do not contribute or learn any information during these updates, making the remaining active neurons to learn and reduce the error. The effect is that the network becomes less sensitive to the specific weights of those neurons. This, in turn, results in a net that is able to improve generalization and is less likely to overlap with training data. In Figure 3.6, it is illustrated, respectively, a neural network without the dropout function and with the dropout function, with a 0.5 adjustment, i.e., a 50% probability.

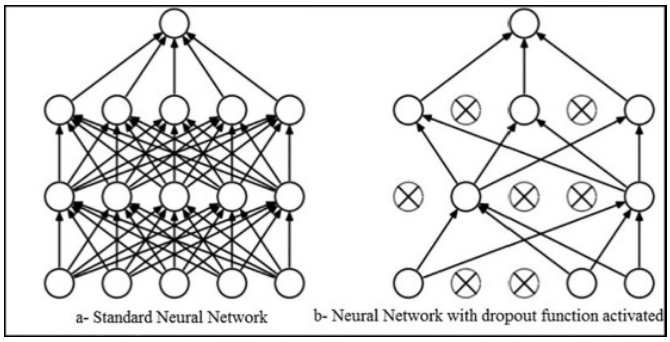


Figure 3.6: Dropout Regularization Technique

3.3 Grad-CAM

In this thesis, we used 1-D Grad-CAM which relies on the same idea depicted as follows. Grad-CAM plays a crucial role in this dissertation since, as explained previously, allows us to gain interpretability of the problem. This technique, will allow the technician to detect at an early state if the motor is functioning in a healthy way or not. This technique, will consist in heatmaps on the frequency domain of multi-variate time series, highlighting the parts that are most important and so, triggering the decision of our 1-D CNN. In other words, with Grad-CAM we will have different clusters on the frequency domain explaining each different faulty state.

Selvaraju et al. suggested utilising Grad-CAM to produce a colour visualisation map for examining the CNN model's region of interest. An enhancement over conventional CAM [Zhou et al. 2016] is grad-CAM [Selvaraju et al. 2017]. This method is challenging to implement in other CNN architectures since CAM demands a CNN built on the global average pooling (GAP) [M. Lin, Chen, and Yan 2013] architecture. The GAP approach, which is intended to alter the fully connected layer, differs from the conventional pooling method in that each 2D feature map is compressed into a 1D vector using an average function before being classified using the softmax function. Figure 3.7 shows the GAP operation, which is stated in Equation 3.9.

$$z_c = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} u_{c(i,j)} \tag{3.9}$$

where z_c is the cth 1D feature after the GAP operation; H and W are the height and width of the 2D feature map, respectively; and u_c is the cth convolved feature map at position (i,j). The CAM architecture is depicted in Figure 3.7. As displayed in Figure 3.7, the attention map of the model is produced from the sum of convolved feature maps and weights that are between the GAP layer and output. The detailed CAM operation is expressed in Equation 3.10.

$$L^c_{x,y} = \sum_k w^c_k A_{k(x,y)} \tag{3.10}$$

where $L^c_{x,y}$ is the class activation map in category c, w^c_k is the weight of the kth feature map, and $A_{k(x,y)}$ is the kth convolved feature map at position (x,y). For overcoming the drawbacks of CAM, Grad-CAM improves the structure of the method in order to generalize various CNNs. First, the output score of each category is calculated using the Equation 3.11

$$S_c = \frac{1}{H \times W} \sum_k w^c_k A_k \tag{3.11}$$

where S_c is the score of the model output in category c; H and W are the height and width of the feature map, respectively; w^c_k is the weight of the kth feature map in category c; and A_k is the kth feature map. The gradient between the output score S_c and

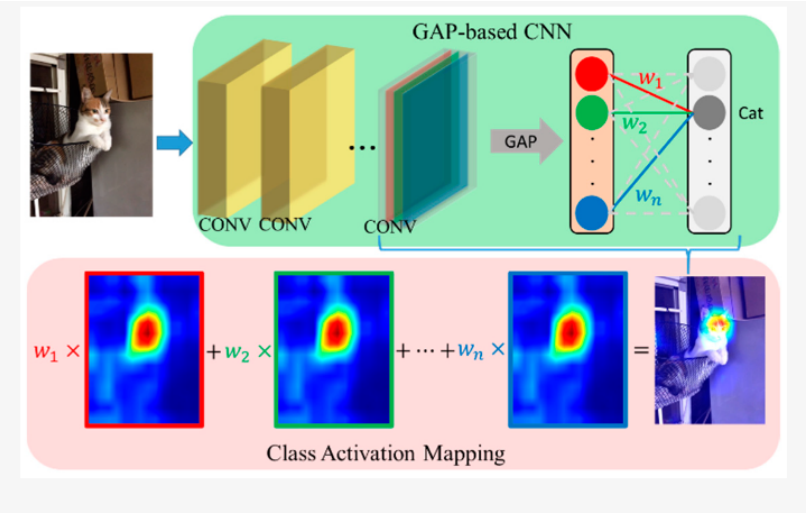


Figure 3.7: CAM architecture

the feature map A_k is calculated using Equation 3.12 and 3.13 to acquire the category discrimination positioning map.

$$\delta_k^c = \frac{\partial S_c}{\partial A_k} \tag{3.12}$$

$$G^c = ReLU(\sum_k \delta_k^c A_k) \tag{3.13}$$

where δ_k^c denotes the weight of the kth feature map and G_c is the normalized heat map of category c. The architecture and examples of Grad-CAM are displayed in Figure 3.8 and Figure 3.9, respectively. The benefit of Grad-CAM is that its operation is based on output results and feature maps. Therefore, the structure of the CNN does not influence the Grad-CAM operation.

The traditional CNN architecture of LeNet-5 mainly uses the convolutional layer to extract the features of the input data and apply the fully connected layer to classify the features. The convolutional layer convolved the input data and convolution kernels with a restricted region of the visual field, known as the receptive field, to automatically extract features without the need for expert experience and knowledge. However, this network architecture has the following shortcomings: (1) it is difficult for users to understand the basis of model feature extraction and (2) the use of a fully connected layer makes the network generate a large number of parameters, which requires a lot of computing time and expensive hardware costs. To improve these problems, the neuro-

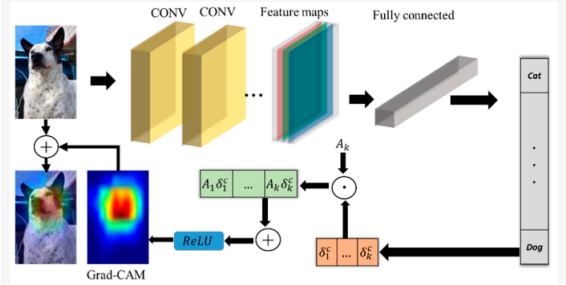


Figure 3.8: Grad-CAM architecture

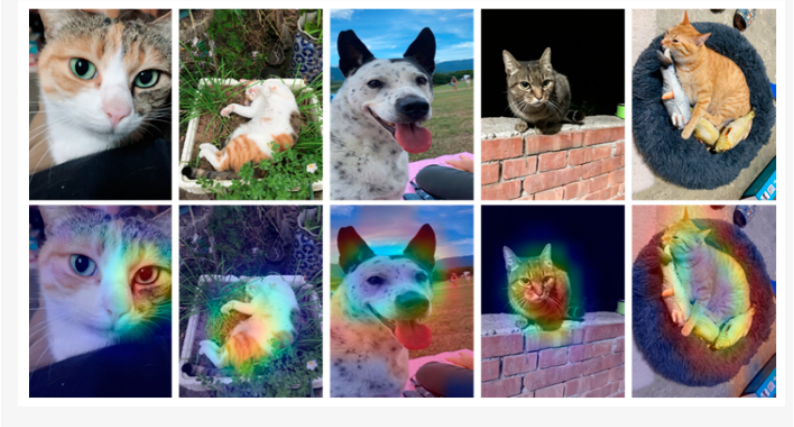


Figure 3.9: Examples of Grad-CAM

fuzzy network was adopted to replace the fully connected layer in the CNN network in this study. It reduces a large number of learnable parameters and improves the classification performance of the network by imitating the fuzzy logic of the human logic mechanism. In addition, by introducing the Grad-CAM method, which can generate a separate visualization image for each class, users can clearly understand the basis of network classification and provide model interpretation.

For further reading please refer to [C.-J. Lin, and Jhang 2021].

Chapter 4

SimuLink and MatLab

In this chapter, we tried simulating the behavior of an induction motor using SimuLink. In order to verify that we successfully simulated this, we made use of two databases: UCC and MAFAULDA.

Please refer to Figure 4.1 to see the structure of our simulation in SimuLink. As we can see from the graph, we simulated a three phase induction motor with squirrel cage whose input is the Hz at which the motor spins. In addition, on the right hand side we can see the outputs generated by this simulation (rotor speed, stator magnetic flux and so forth). The simulation usually was executed for 2 seconds obtaining around 25k data point for each output.

4.1 UCC database

This database is made of data from three induction motors located at University College of Cork. Referring to Figure 4.2, 4.3 and 4.4, it can be seen the metadata of these three induction motors. This metadata contains input and output spinning frequencies, voltage applied and so forth

Firstly, as this data seemed to be noisy, we applied Fast Fourier Transformation to smooth it. The data was smoothed taking the highest amplitude in the frequency spectrum domain and making the remaining amplitudes equal to 0. Afterwards, it was applied Inverse Fast Fourier Transformation to get back to a curve in the time domain. To gain a better understanding of this process, please refer to Figure 4.5 and 4.6, these contain this process for a single data point for Magnetic Flux - Grundfos. In the first of these pictures, it can be seen on the left one the frequency domain for a single data point, in the middle photo the highest frequency highlighted and on the right one, the Inverse FFT curve versus the raw data. Moreover, in Figure 4.6 we zoom in to see better the smoothed curve (in blue) versus the raw data (in light gray).

Then, the next step was to validate that this process was properly implemented by comparing the smoothed curve we obtained previously with the data we exported from our simulation in MatLab. The comparison was performed using **dynamic time**

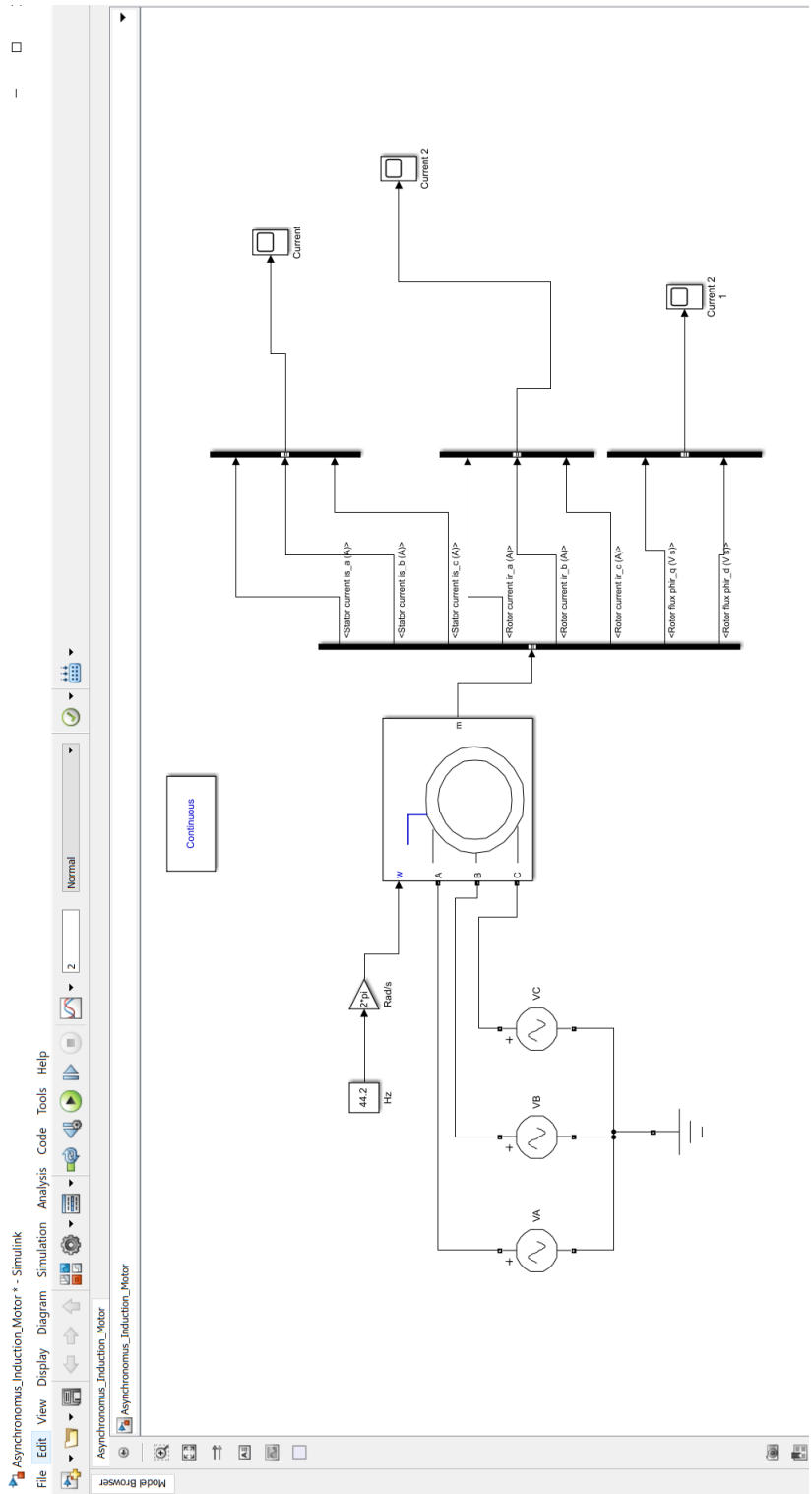


Figure 4.1: Simulation of a three phase induction motor's behavior using SimuLink

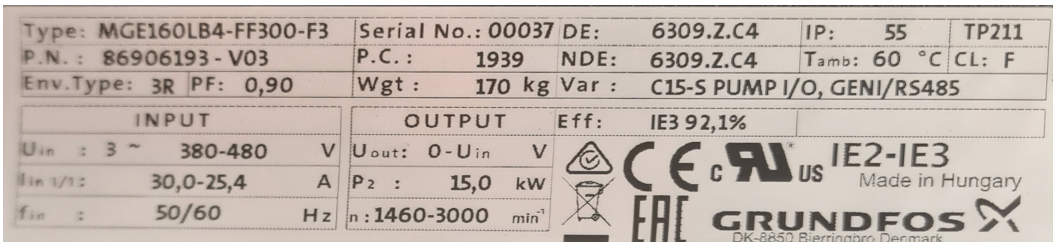


Figure 4.2: Metadata of first motor

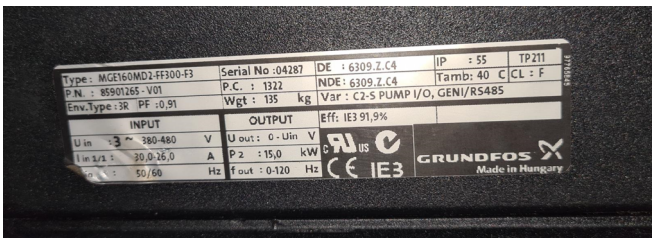


Figure 4.3: Metadata of second motor

Figure 4.4: Metadata of third motor

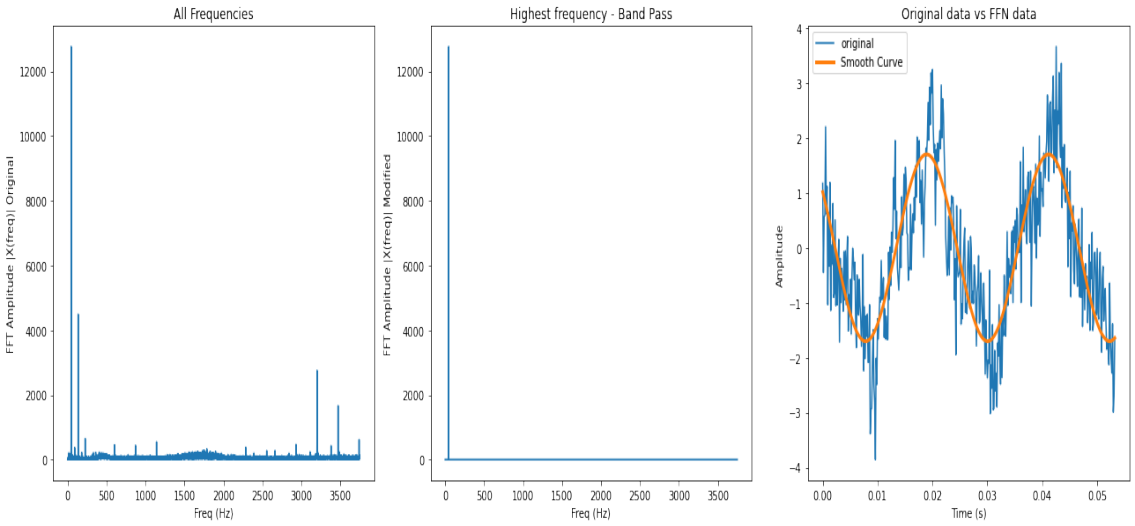


Figure 4.5: Three steps process

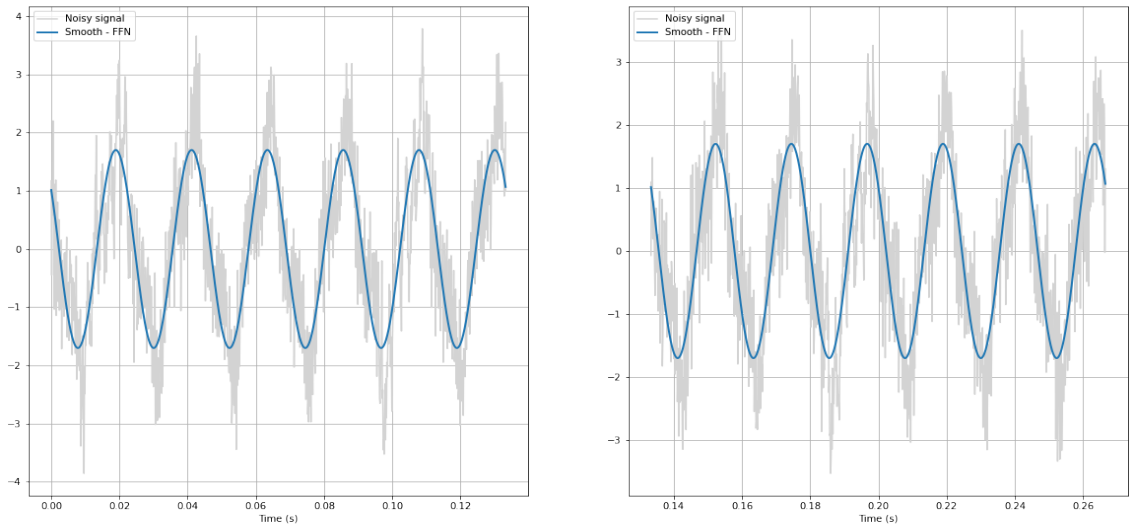


Figure 4.6: Raw data vs Smoothed data

warping, which is a technique that compares the similarity between two time series. In order to make a good comparison, the simulation in Matlab was run at different input Hz. Referring to Figure 4.7, we compared the distance obtained with respect to our smoothed data, with the mean of the simulated data (in order to have a comparison). Moreover, from this table we can see that the smallest distances arise when the motor spins at 50Hz.

However, this distance can be improved (i.e. can be smaller). We used Curve Fitting in Matlab to fit a regression of sum of Sines. As FFT and this regression of sum of Sines are pretty much the same techniques, we can compare parameters obtained from Curve Fitting and FFT. Figure 4.8 refers to the results obtained at different initial configurations of the simulation. It turned out that, after hyperparametrizing the parameters, the most similar simulation with respect to the raw data and smooth data was when the motor spun at 47.5Hz and initial angles of (38,-82,158) (i.e. the three phase of the induction motor).

In addition, other advantage that Curve Fitting provides is that it gives statistical insight. Referring to Figure 4.9, in this scenario we can see that our smoothed data approximately explains 70.82% of the variability, which is great since we can perfectly say that approximately 30% is noise.

Then, we calculated the dynamic time warping for this best configuration of the simulation. Figure 4.10 shows this distance, turning out that the distance was significantly reduced in comparison with the previous best configuration.

Finally, this study and methodology were applied to all the different data points for each machine and measure. The results obtained were good and approximately always we explained around 70-80% of the variability of the signal involved. To end up, as these results were considerably good we tried implementing this approach with another induction motor database: MAFAULDA.

Signal: Magnetic Flux
Machine: GrundFoss

	50 Hz		55 Hz		60 Hz	
	Short Period 0.1s	2 seconds	Short Period 0.1s	2 seconds	Short Period 0.1s	2 seconds
Mean of Flux – 1st Signal	**20.15**	**62.91**	20.44	63.98	22.91	71.82
Smooth 1st Signal	**9.06**	**21.97**	11.95	33.65	15.14	44.06
Mean of Flux – 2nd Signal	**20.17**	**63.03**	20.38	63.99	22.78	71.81
Smooth 2nd Signal	**9.00**	**22.25**	11.61	34.27	14.73	44.03

Dynamic time warping (distance) between data generated through Matlab at different Hz and Smoothed signals from original data.

Figure 4.7: Dynamic time warping between smooth data and data generated in Matlab at different frequency inputs

CURVE FITTING

Sum of Sine Curve Fitting: $a * \sin(b * x + c)$

Summary	a	b	c
Time vs Smoothed Flux (untitled fit 1)	1.701	282.7	2.502
Time vs Raw Data (untitled fit 3)	1.703	282.8	2.431
Time vs Values Matlab 1 50Hz (untitled fit 6)	1.701	314.2	3.137
Time vs Values Matlab 2 50 Hz (untitled fit 7)	1.701	314.2	1.566
Time vs Values Matlab 1 47.5 Hz (38,-82,158) (untitled fit 8)	1.701	282.7	2.517
Time vs Values Matlab 2 47.5 Hz (38,-82,158) (untitled fit 9)	1.701	282.7	0.9459

Figure 4.8: Making use of Curve Fitting to tuned the parameters of our simulation.

Goodness of fit	SSE	R-Square	Adj R-Square	RMSE
Time vs Smoothed Flux	3.22e-17	1	1	4.635e-11
Time vs Raw Data	8958	0.7083	0.7082	0.7729
Time vs Values Matlab 1 50 Hz	0.1686	1	1	0.007845
Time vs Values Matlab 2 50 Hz	0.1685	1	1	0.007844
Time vs Values Matlab 1 47.5 Hz (38,-82,158)	0.1329	1	1	0.007079
Time vs Values Matlab 1 47.5 Hz (38,-82,158)	0.1317	1	1	0.007048

Figure 4.9: Statistical insight

Dynamic time warping for 47.5HZ and (38,-82,158)

Signal: Magnetic Flux
Machine: GrundFoss

	50 Hz		47.5HZ and (38,-82,158)	
	Short Period 0.1s	2 seconds	Short Period 0.1s	2 seconds
Mean of Flux – 1st Signal	20.15	62.91	**19.81**	**62.03**
Smooth 1st Signal	9.06	21.97	**2.82**	**8.69**
Mean of Flux – 2nd Signal	20.17	63.03	**19.79**	**61.93**
Smooth 2nd Signal	9.00	22.25	**4.05**	**9.75**

Note: it has been a reduction in the distance. We use sine regression in order the better fit the parameters of the motor.

Figure 4.10: Best configuration vs previous configuration

4.2 MAFAULDA database

For this particular database it was studied and followed the same steps as in the previous section. For instance, refer to Figure 4.11 and 4.12 to see how FFT along with Band-Pass were applied on a data point that functions healthily.

However, from this figure, we can see that the smoothed data is not as accurate as in the previous section. Meaning that, probably our smoothed data losses to much information from the raw that, which is not acceptable.

Therefore, in order to verify this possibility mathematically, we calculated the RSS associated with raw data and regression of sum of Sines. From Figure 4.13, it can be seen that our thought was right and indeed, we lose approximately 50% of the information from the raw data if we apply FFT and its inverse.

Afterwards, we realised that probably this was because our selection of BandPass was too extreme and it might not have happened if we had not set that amount of frequencies equal to 0 before applying FFT inverse. Thus, we made use of Curve Fitting in order to extend the number of terms involved in the regression of sum of Sines. Referring to Figure 4.14, we can see that RSS seems to not get better than 60%, and so, we can not move forward assuming that 40% of the data is merely noise. What is more, RSS seemed to get worse by giving to the regression more terms.

Unfortunately, due to the difficulty of simulating accurately and obtaining the desired outputs of an induction motor using SimuLink we decided, after finding these results, that the most appropriate thing was to **change the scope of the thesis**.

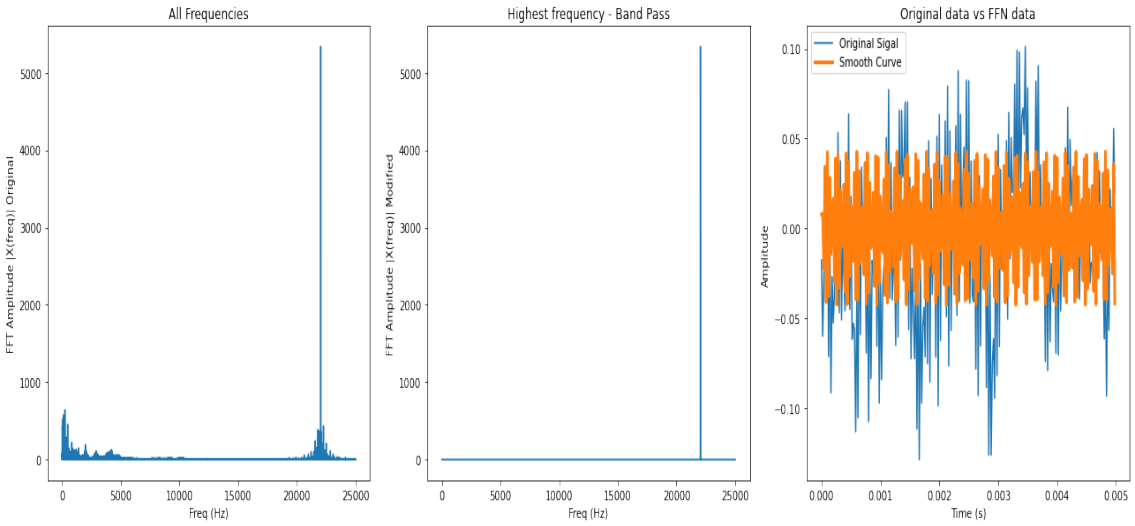


Figure 4.11: Three steps process

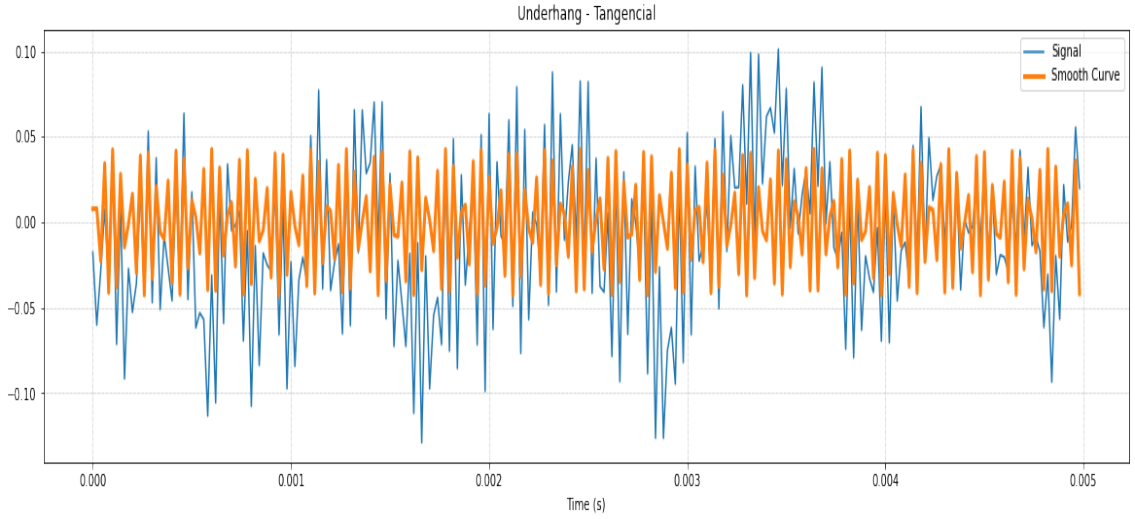


Figure 4.12: Raw data vs Smoothed data

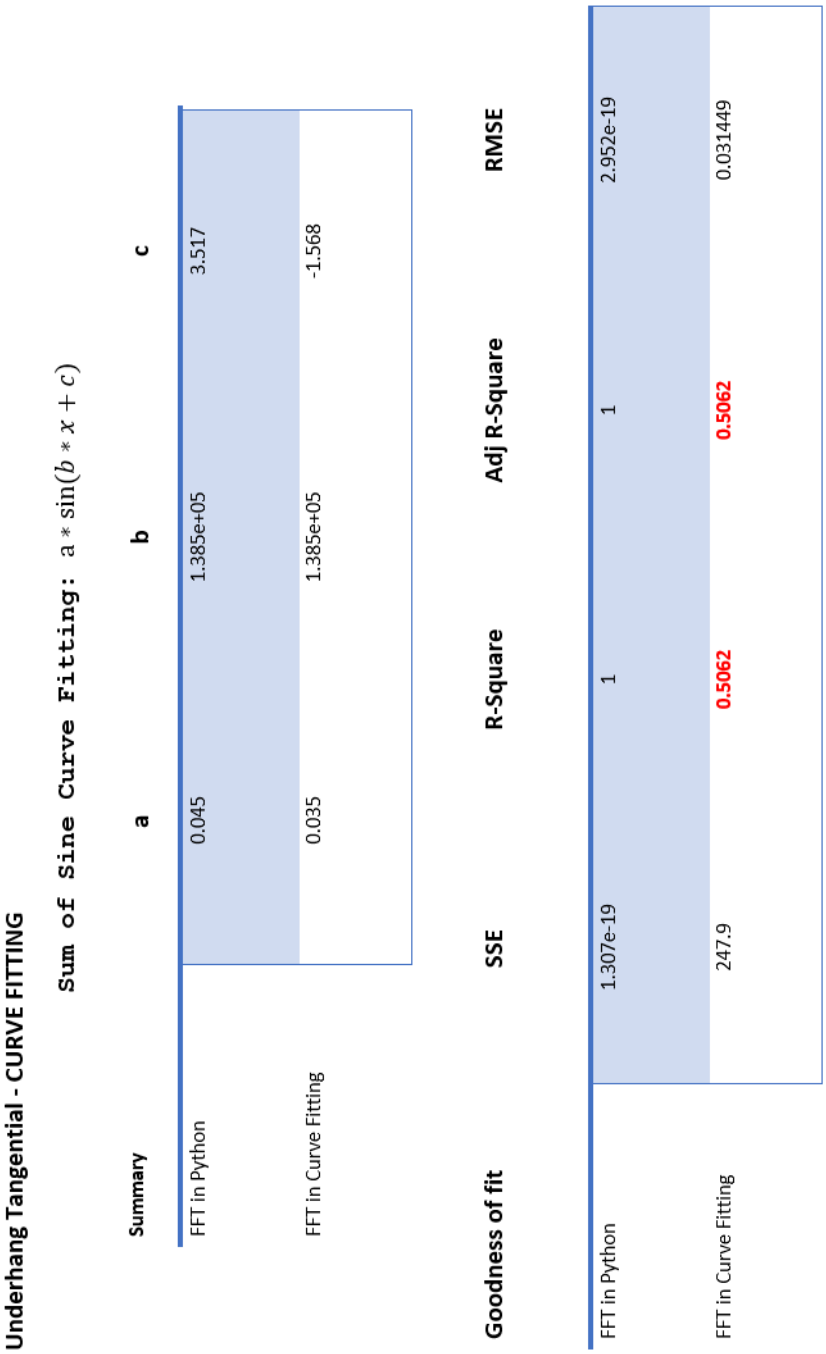
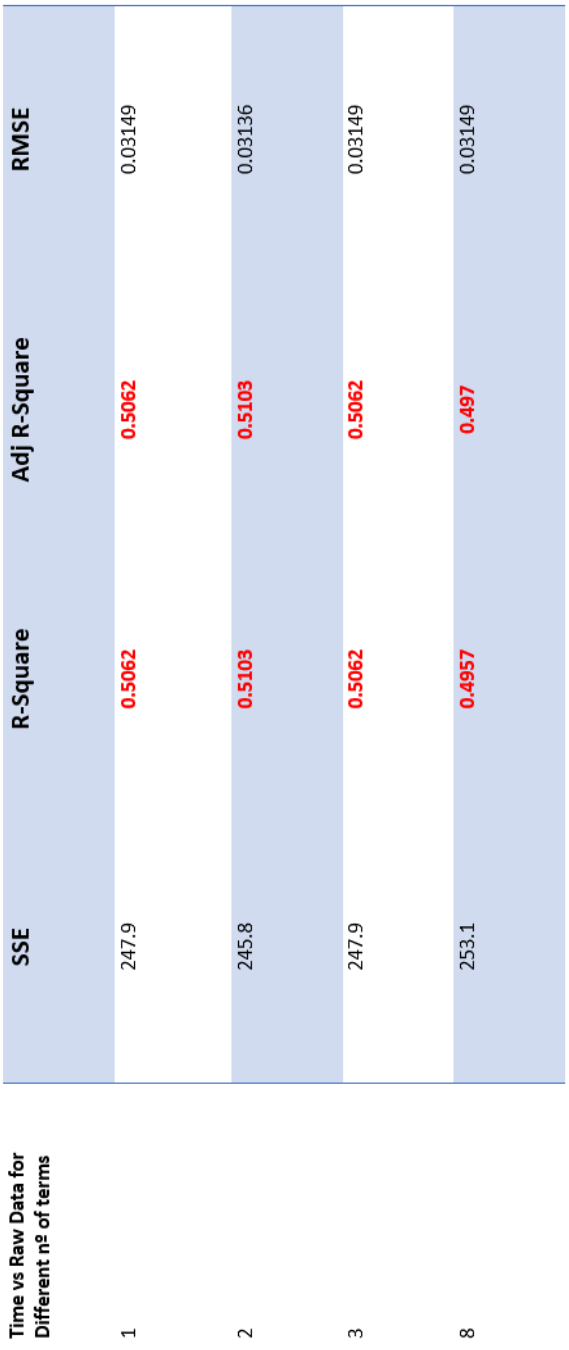


Figure 4.13: Statistical insights for a normal functioning data point from MAFAULDA



Time vs Raw Data for Different nº of terms	SSE	R-Square	Adj R-Square	RMSE
1	247.9	0.5062	0.5062	0.03149
2	245.8	0.5103	0.5103	0.03136
3	247.9	0.5062	0.5062	0.03149
8	253.1	0.4957	0.497	0.03149

Figure 4.14: Statistical insights for a normal functioning data point from MAFAULDA for more terms

Chapter 5

Methodology for 2nd Approach

In this chapter we will present two main topics: the preprocessing needed in order to handle and study MAFAULDA database and different CNNs that performed very accurately and reliably.

5.1 Preprocessing Techniques

Jumping back to how the Machine Fault Simulator captures the vibration of the induction motor, it has two accelerometers allocated on the underhang and overhang bearings. In Figure 5.1 we can see a plot with all the different components that the Simulator contains [Alzghoul et al. 2021].

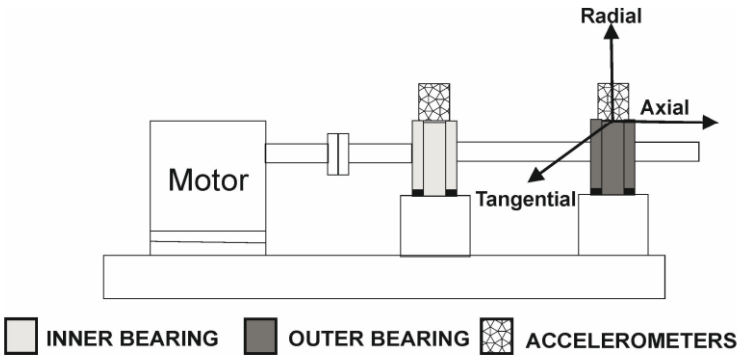


Figure 5.1: Schematic configuration of a machine fault simulator

MAFAULDA database has data points for different states of the motor: normal function, imbalance fault, horizontal and vertical misalignment faults and, inner and

outer bearing faults. So that, as each sequence was generated at 50kHZ within a 5 seconds period, in each data point we have 250000 samples. To give a better insight of what consists MAFAULDA database:

- Normal condition: the ideal condition where there is no faults or defects in the bearings and motor shaft. It includes a total of 49 data samples with a rotating speed ranging from 12.288 Hz to 61.44 Hz.
- Imbalance: scenarios of the imbalance state was created using different load values. The loads are 6 g, 10 g, 15 g, 20 g, 25 g, 30 g and 35 g. Every load has between 44 to 49 data sets, with a total of 333 data samples for the whole class.
- Horizontal misalignment: misalignment was made by shifting the motor shaft horizontally by a fixed degree of displacement; 0.5 mm, 1.0 mm, 1.5 mm and 2.0 for multiple rotational speeds, making a total of 197 data samples.
- Vertical misalignment: misalignment was done by shifting the motor shaft vertically by a fixed degree of displacement; 0.51 mm, 0.63 mm, 1.27 mm, 1.40 mm, 1.78 mm, 1.90 for various rotational speeds, making a total of 301 data samples.
- Bearing faults: both lower and upper bearings are subjected to three types of faults (retainer, outer race, and ball). Two Bearings are placed in two distinct locations underhang and overhang. Underhang is referred to the position between the rotor and motor, and overhang is located at the outer most position after the rotor. Three unbalancing load values of 6 g, 20 g and 35 g were simulated for all faults, leading to 558 underhang and 513 overhang data samples.

Therefore, as we aim to have a good explainability of the problem through the visualization of a heatmap on the frequency domain of multi-variate time series, FFT was used to convert our input from the time domain to the vibration signal frequency domain. In addition, as the dimensions of a single data point are excessive and difficult to handle without running out of memory RAM (under our available resource), we need to apply a dimensionality reduction technique (it was aimed to reduce the input size from 250000 to 5000). In this particular paper, it was used Rapid Fast Fourier Transformation Chapter 3.1 (rfft in Python) in order to obtain the frequency domain spectrum of each data point. In addition, should be highlighted that we just only focused on the two accelerometers and their 3 directions: axial, radial and tangential. So that, our input size will consist in 1951 data points of dimensions 5000 x 6.

In this thesis also, we will consider **two experiments**:

- Considering 6 different failures, without regarding different severity in each class.
- As our results, when it comes to interpretability, with underhang failure were not appropriate, we considered a 4 type of failures experiment that consists in: normal function and underhang failures (but taking into consideration different causes of this failure i.e. cage fault, ball fault and outer race)

Moreover, after having applied rfft on each different signal, we obtain the amplitude of each frequency (in absolute value). So that, as this scale might be different for each data point, we deal with this problem applying **MinMaxScaler in a column wise way** for each direction of each accelerometer. With the use of this scaling technique, we ensure that all these frequency domain spectrum's amplitudes will be between 0 and 1 (also it is useful when it comes to select a proper Activation Function). Referring to Figure 5.2, we can see how a single data point after applying FFT would look like (this point was taken from a normal functioning of the motor)

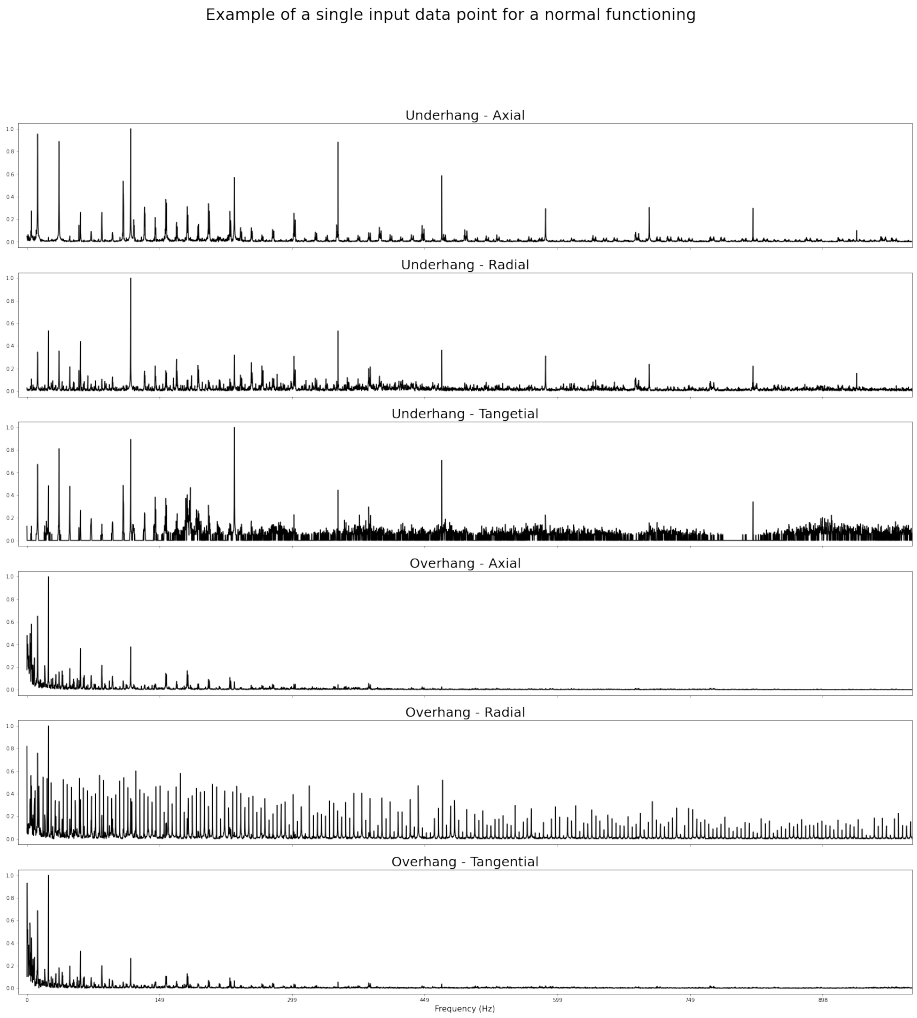


Figure 5.2: Frequency spectrum example for a sample point

Furthermore, as we can see from Figure 5.2 the x-axis is shared and constant for all these 6 different directions of the two different accelerometers. So that, in our CNN input we will consider just only each y-axis (i.e. amplitude).

Consequently, as we know the labels, this experiment will be a supervised-learning. Thus, it was created a y-vector containing all the labels for the two different experiments.

1st Experiement Y Size: (1951, 6)
2nd Experiement Y Size: (1951, 4)

To sum up with all the preprocessing explained, for each data point out 1951 different data points, it was applied Rapid Fast Fourier Transformation in each accelerometer's signal in order to reduce the massive dimensionality of the database from 250000 to 5000. Obtaining as a result an input size of:

Input Size: (1951, 5000, 6)

Lastly, as in every Machine Learning Model, the data has to be split up between Training and Validation sets. In particular in this thesis, it was always split the data into 70%-30%. Thus, the resulting dimensions for Xtrain, Xtest, ytrain and ytest are as follows:

X - Training size: (1560, 5000, 6)
X - Test size: (391, 5000, 6)
y - Training size: (1560, 6)
y - Test size: (391, 6)

Noticing that if we were dealing with the second experiment the y vector would have these dimensions:

y - Training size: (1560, 4)
y - Test size: (391, 4)

5.2 Deep Learning Model: CNN

The proposed Deep learning model is 1-D Convolutional Neural Network (CNN), referring to [Wibawa et al. 2022] CNN skillfully addresses time-series issues. The results of recent experiments using CNN for time-series forecasting tasks, primarily involving financial data, are encouraging. CNN decodes raw pixel data to derive picture characteristics [Jogin et al. 2018]. However, due of the numerical pattern, the raw data extraction is not required in time-series analysis. Compared to other algorithms like RNN, GRU, and LSTM, CNN can train models twice as quickly and boost accuracy by up to 30% [Rajagukguk, Ramadhan, and Lee 2020]. Reducing the number of parameters can improve the effectiveness of model learning [Qin, Yu, and Zhao 2018]. Because CNN enables dilated convolutions, in which filters can be used to compute cell dilations, it is useful for forecasting time-series data. The neural network can comprehend the relationships between the various observations in the time-series better thanks to the magnitude of the gap between each cell [Borovykh, Bohte, and Oosterlee 2018].

In our experiment we are dealing with multi-variate time series. Thus, the most appropriate approach is to apply 1-Dimensional Convolutional Neural Network, along with 1-D MaxPooling, Dropout and so forth. In addition, since we are in a supervised classification scenario, our loss function is *Sparse Categorical Cross Entropy* and because of, all our data points' values are between 0 and 1 we will use *ReLU* as Activation Function for each convolutional layer expect for the last Dense layer. Lastly, it was applied in each convolutional layer L_1 and L_2, bias and activity regulizers in order to avoid the possible overfitting of this network [*Layer weight regularizers* n.d.]

5.2.1 Non-tuned CNN

When we very first built the configuration of the CNN, it turned out to be very successful and accurate. This configuration is as follows in Figure 5.3 and Figure 5.4 (using Keras Sequential). Moreover, in order to classify with a CNN we need a Dense Layer along with SoftMax Function in order to get the probability of a particular sample point to be classified in each class. Should be highlighted that in the first experiment the number of neurons in the last Dense layer is 6 and in the second experiment is 4 neurons.

The different hyper parameters that this particular CNN has are: Dropout - 0.1 ; Kernel size - 13; Pool size - 2; input shape = (5000 ,6). Plus, we added callbacks: one early stopping callback that will stop if the validation accuracy does not get better within 5 epochs and one in charge of saving up the best model to then, save its weights and its configuration to load it later without needing to compile the model again. Lastly, the optimizer used was Adam with the default learning rate (it will be tuned in 5.2.2) .

5.2.2 Hyperparametrizing CNN

In this section we focus on the first experiment, as the second experiment reached a perfect accuracy (later explained in). The first experiment was enough accurate, only

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d_20 (Conv1D)          (None, 5000, 16)          1264

 batch_normalization_8 (Batc  (None, 5000, 16)         64
 hNormalization)

 dropout_8 (Dropout)         (None, 5000, 16)          0

 conv1d_21 (Conv1D)          (None, 5000, 32)          6688

 max_pooling1d_16 (MaxPoolin  (None, 2500, 32)         0
 g1D)

 batch_normalization_9 (Batc  (None, 2500, 32)         128
 hNormalization)

 conv1d_22 (Conv1D)          (None, 2500, 128)         53376

 max_pooling1d_17 (MaxPoolin  (None, 1250, 128)        0
 g1D)

 dropout_9 (Dropout)         (None, 1250, 128)         0

 conv1d_23 (Conv1D)          (None, 1250, 32)          53280

 max_pooling1d_18 (MaxPoolin  (None, 625, 32)          0
 g1D)

 conv1d_24 (Conv1D)          (None, 625, 16)           6672

 max_pooling1d_19 (MaxPoolin  (None, 312, 16)          0
 g1D)

 flatten_4 (Flatten)         (None, 4992)              0

 dense_4 (Dense)             (None, 6)                 29958

 activation_4 (Activation)   (None, 6)                 0

=================================================================
Total params: 151,430
Trainable params: 151,334
Non-trainable params: 96
_________________________________________________________________
```

Figure 5.3: Configuration of the CNN

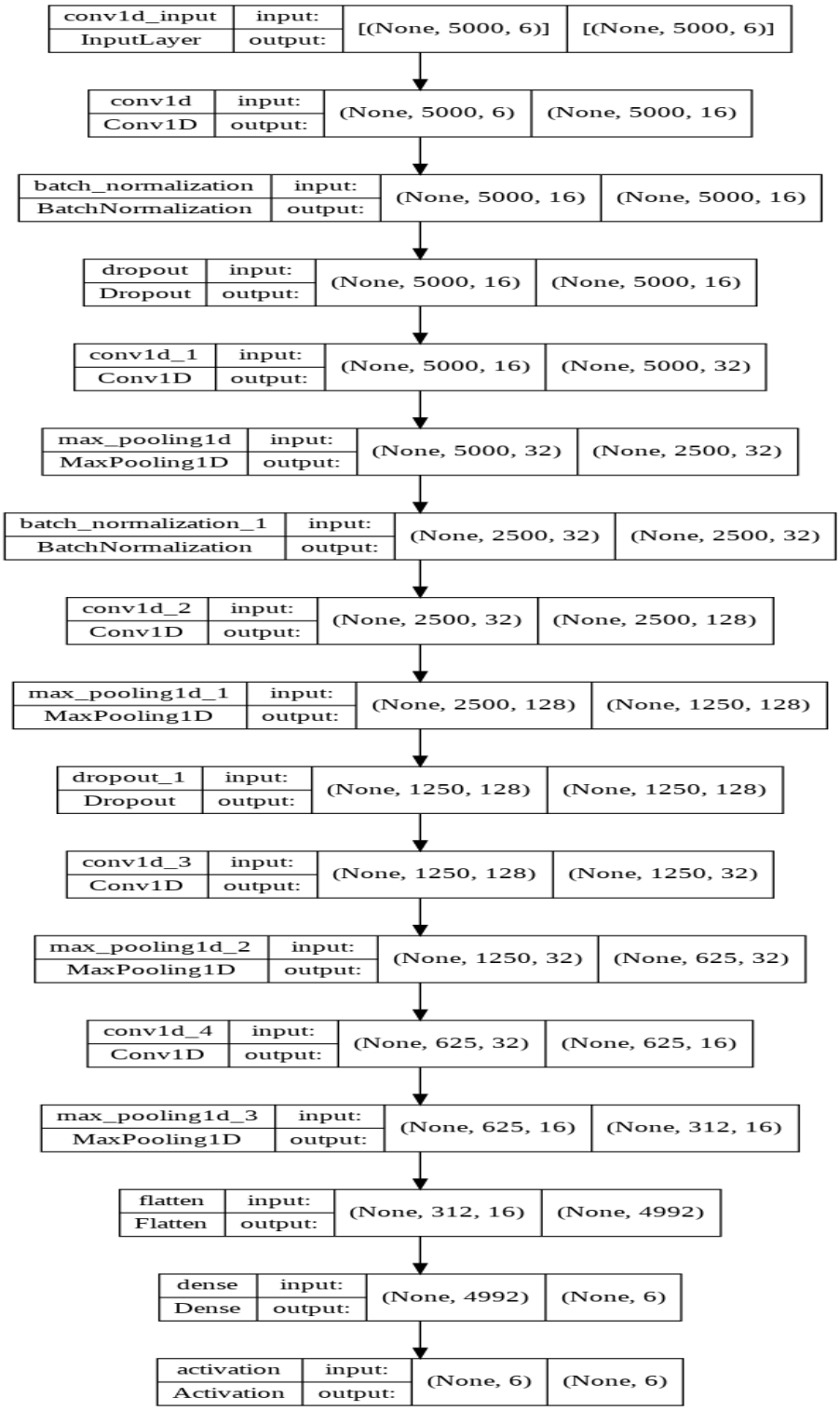


Figure 5.4: Configuration of the CNN

misclassifying an unique and single data point. But, for the sake of the experiment, we used Talos [*Talos - Hyperparameter Optimization for TensorFlow, Keras and PyTorch* n.d.] in order to tune the hyperparameters of our CNN to get an even better accuracy.

We created a dictionary in Python containing all the range of possible values for the hyperparameters. Refer to Figure 5.5 to see which values were used.

```
# set the parameter space boundary
p = {'activation':['relu'],
     'optimizer': ['Adam'],
     'kernel_size_1':[7, 9, 11, 13],
     'kernel_size_2':[7, 9, 11, 13],
     'kernel_size_3':[7, 9, 11, 13],
     'learning_rate' : [1e-03, 1e-02, 1e-04, 1e-05],
     'pool_size' : [2,3,4],
     'padding': ['same'],
     'strides': [1],
     'filters_1': [16, 32, 64, 128],
     'filters_2': [16, 32, 64, 128],
     'filters_3': [16, 32, 64, 128],
     'dropout': [.01, 0.1, 0.05],
     'batch_size': [8]}
```

Figure 5.5: Range of Values for Hyperparametes

After running this hyperparametrizing technique, the parameters that turned out to be outperforming the previous ones mentioned in the non-tuned CNN were: kernel size 1 - 11; kernel size 2 - 13; kernel size 3 - 11; learning rate - 0.001; pool size - 4; filter 1 and 2 - 64; filter 3 - 16; dropout - 0.01.

Chapter 6

Experimental Results

In this section it will be presented the results for the two experiments undertaken. As a quick reminder, these two experiments are:

- Considering 6 different states of the motor, without regarding different severity in each class.
- As our results, when it comes to interpretability, with Underhang Failure were not appropiate, we considered a 4 type of failures experiment that consists in: normal function and Underhang failures (but taking into consideration different causes of this failure i.e. cage fault, ball fault and outer race)

6.1 Results for 1st Experiment

The model presented in Figure 5.3, was executed for 100 epochs with a batch size equal to 8 and the two callbacks mentioned earlier (early stopping and save best model).

The evolution of its training and validation loss can be seen in Figure 6.1, whereas, the evolution of its training and validation accuracy can be seen in Figure 6.2. We can see from both, that the model converges quickly and it reaches a high accuracy from epoch 15, making it a reliable model since it does not present a raised number of spikes. What's more, it was executed for 100 epochs only, since it seemed to not get better and not improving an accuracy of 99.7% on the validation set and 100% on the training set. A summary of minimum loss function and maximum accuracy reached is as follows:

- The maximum accuracy training found is: 100.0 % and was found in epoch number: 22
- The maximum accuracy test found is: 99.74 % and was found in epoch number: 17

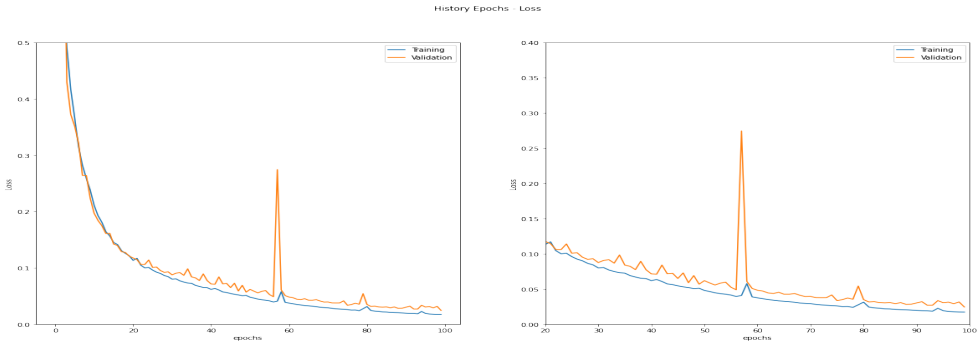


Figure 6.1: Loss Function - epochs from CNN

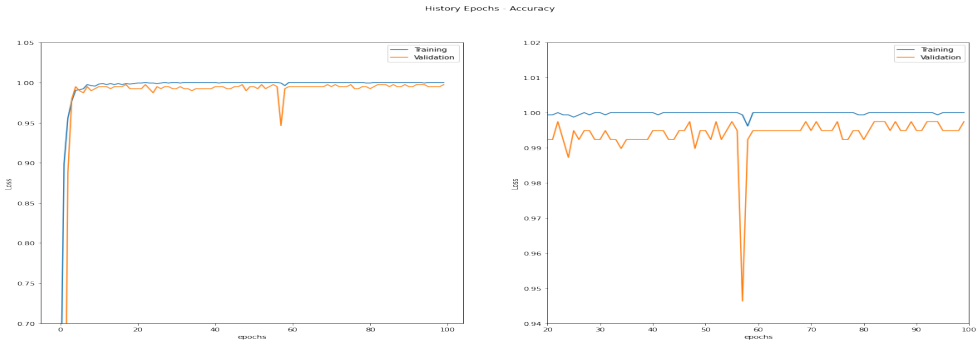


Figure 6.2: Accuracy - epochs from CNN

Moreover, referring to Figure 6.3 it can be seen the Test test Confusion Matrix containing Actual Values vs Predicted Values, noticing only an unique value misclassified. This particular value was classified as Vertical Misalignment when it was in reality an Imbalanced data point. This Confusion Matrix was built using Seaborn library for an outstanding visualization. Moreover, the classification accuracy per class was:

- The accuracy for class Normal is: 100.0% = 10 / 10
- The accuracy for class Hor Mis is: 100.0% = 39 / 39
- The accuracy for class Imbalance is: 100.0% = 67 / 67
- The accuracy for class Ver Mis is: 98.33% = 59 / 60
- The accuracy for class Overhang is: 100.0% = 103 / 103
- The accuracy for class Underhang is: 100.0% = 112 / 112

6.1.1 Interpretability of CNN: Grad-CAM

The performance of the first experiment was good achieving an outstanding accuracy and a small loss value. However, it lacks of interpretability when it comes to allocate the different zones of the spectrum of each data point's signal that triggers our CNN's decision.

Thus, it was applied a Grad-CAM to gain a better understanding of which vibration's frequencies are important on each type of class. This result is crucial, since we can have a visualization on the frequency spectrum that will say if the motor is about to breakdown and the future technician could prevent it at an early stage.

After applying Grad-CAM, we obtained a heat map of each single data-point and then, it was calculated an average heat map for each different class. Refer to Figure 6.5 to see the average heat map. This graph is so helpful since, from it we can see how for instance, vertical misalignment failures mainly focus on a vibration frequencies less than 100 Hz. In addition, other class that should be highlighted is Imbalance because of its clear evidence of which frequency zone of the spectrum is taken into account in order to classify a single data point to this class. On the other hand, class Underhang is not clearly defined and so, this was the result that caused a further investigation: 2nd Experiment.

On top of that, Figure 6.6 contains the average frequency domain spectrum per class. We can notice from this graph and Figure 6.5 how the zones match and so, it makes sense all these results obtained.

Lastly, we will focus on the data point belonging to Vertical Misalignment and classified as Imbalanced. As follows, in Figure 6.4 appears the heat map for this particular data point. We can see from this graph that it was classified as Imbalance since, its

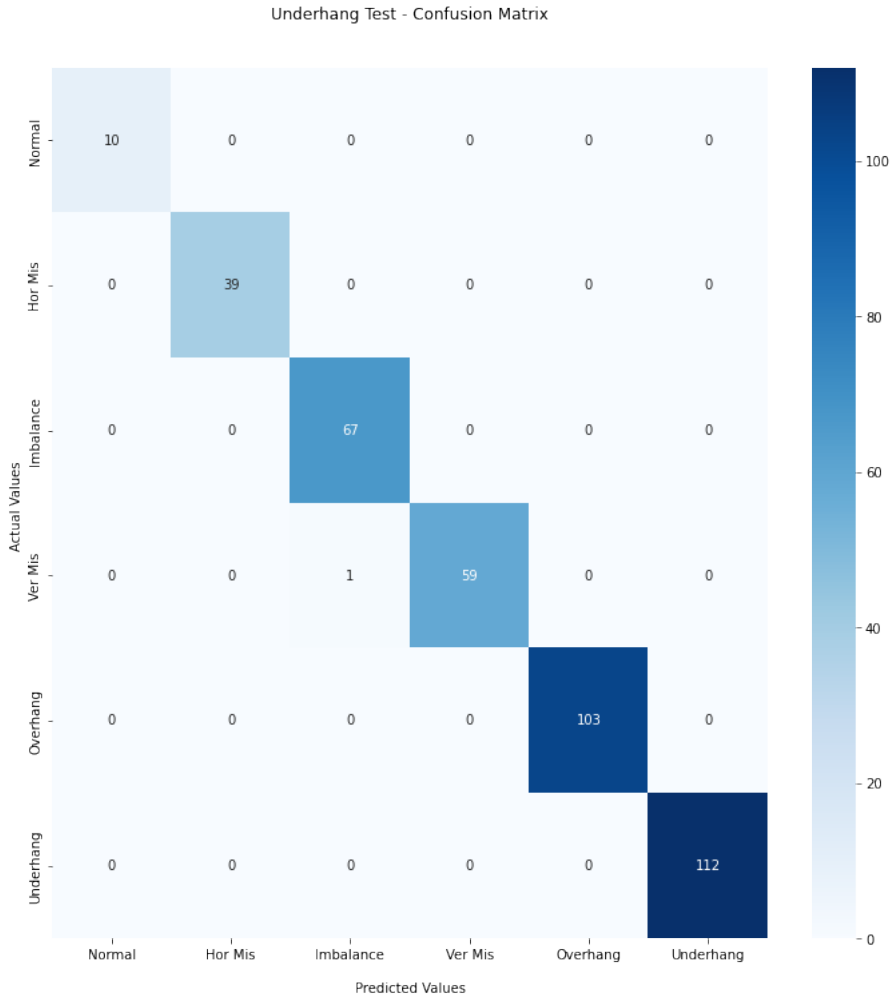


Figure 6.3: Confusion Matrix for the Test Set

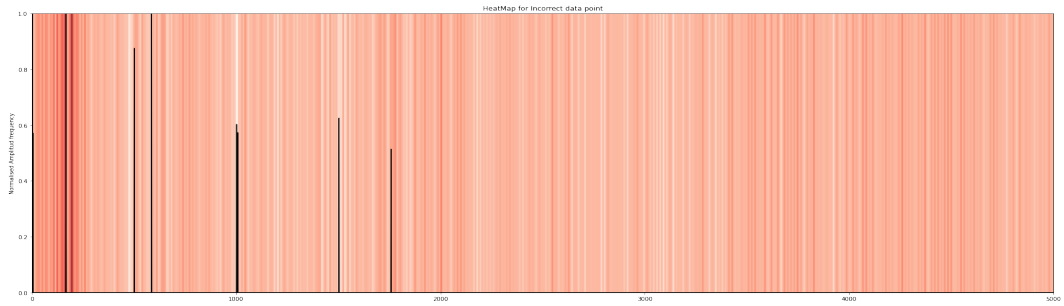


Figure 6.4: HeatMap for the misclassified data point

heat map looks more alike to the average Imbalance heat map more than Vertical Misalignment heat map. So that, this case might have been labelled incorrectly. Also, note the nuance that for this graph there are more red parts as it is not an average of the class, and so, with a single data point we can not highlight that much the part that the CNN takes into consideration.

6.1.2 Hyperparametrizing using Talos on CNN

As previously mentioned, it was executed the same configuration of the CNN Figure 5.3 but with different hyperparameters each time, Figure 5.5. Should be highlighted that there were executed a total of 200 combination of these parameters (using round limit). Moreover, the parameters and results for these 200 combinations were stored into a .csv that then, will allow us to detect and load the best model and its weights.

Having selected and identified the best model, we found that there was a model (5.2.2 to see its hyperparameters) with a **Test Accuracy of 100%** but, its **Train Accuracy dropped to 99.4%**. Thus, I would suggest that tuning this CNN, under this scenario, would not give a statistically significant difference in terms of accuracy. In particular, the test accuracy was 100%, however, the training accuracy dropped and so that, this model did not outperform the previous non-tuned one. Referring to Figure 6.7 and Figure 6.8, it can be seen the Confusion Matrices for the Train and Test sets respectively.

As the Test Accuracy was 100%, we will quote the classification rate per class just only for the Training Set. This is as follows:

- The accuracy for class Normal is: 100.0% = 39 / 39
- The accuracy for class Hor Mis is: 96.84% = 153 / 158
- The accuracy for class Imbalance is: 98.5% = 262 / 266

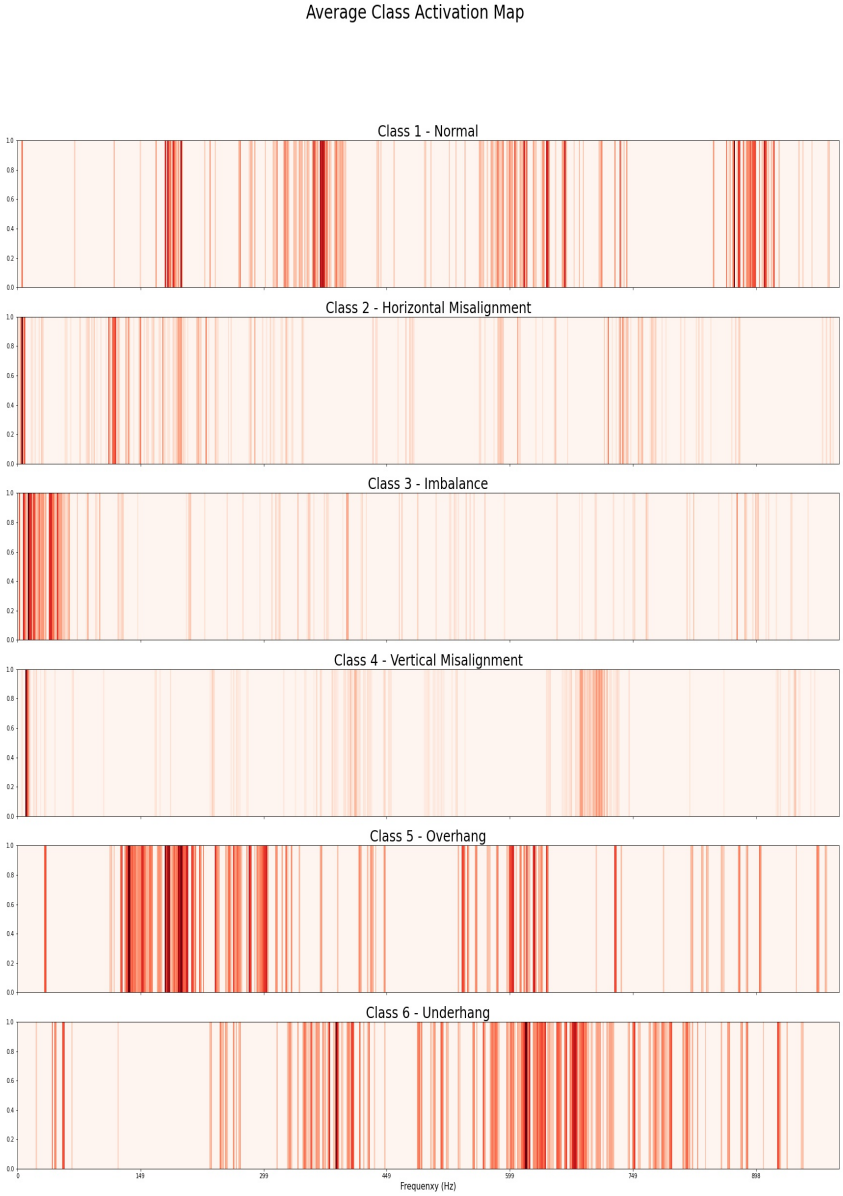


Figure 6.5: Average HeatMap for each class

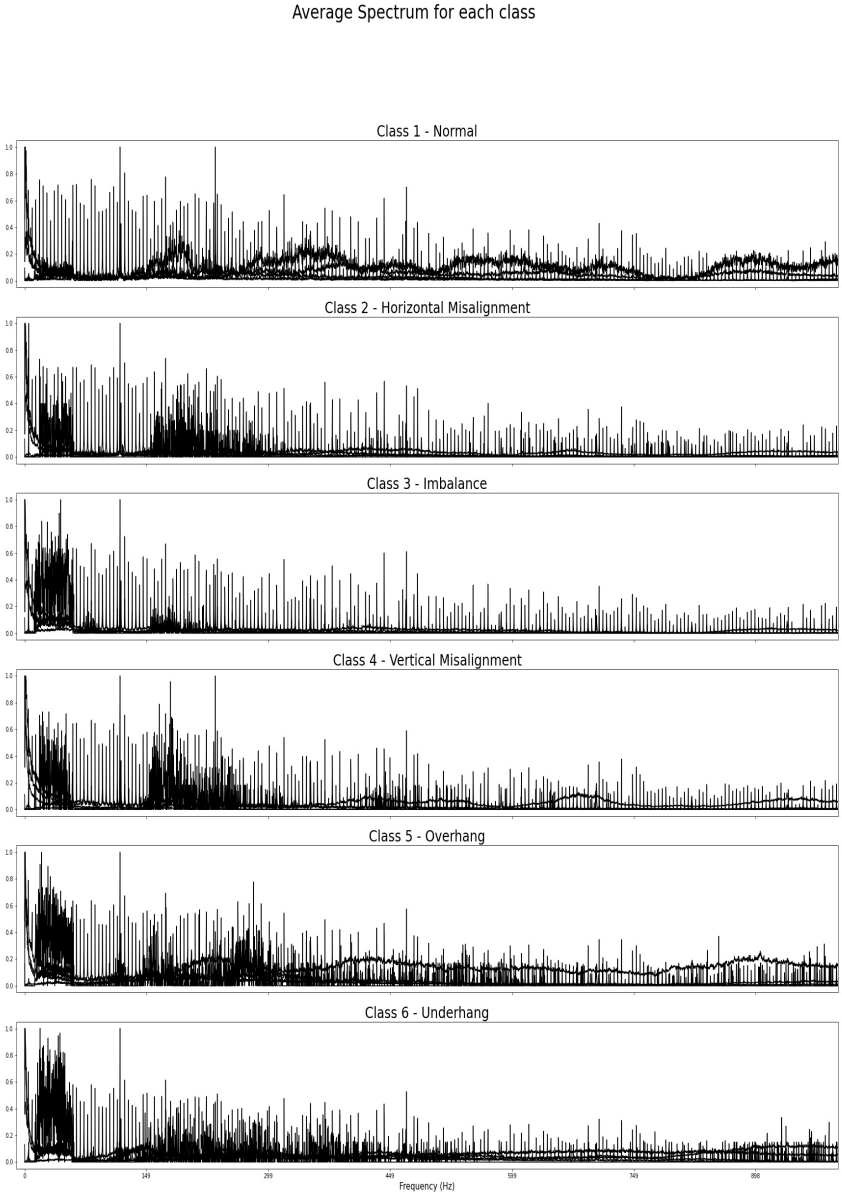


Figure 6.6: Average Frequency Spectrum for each class

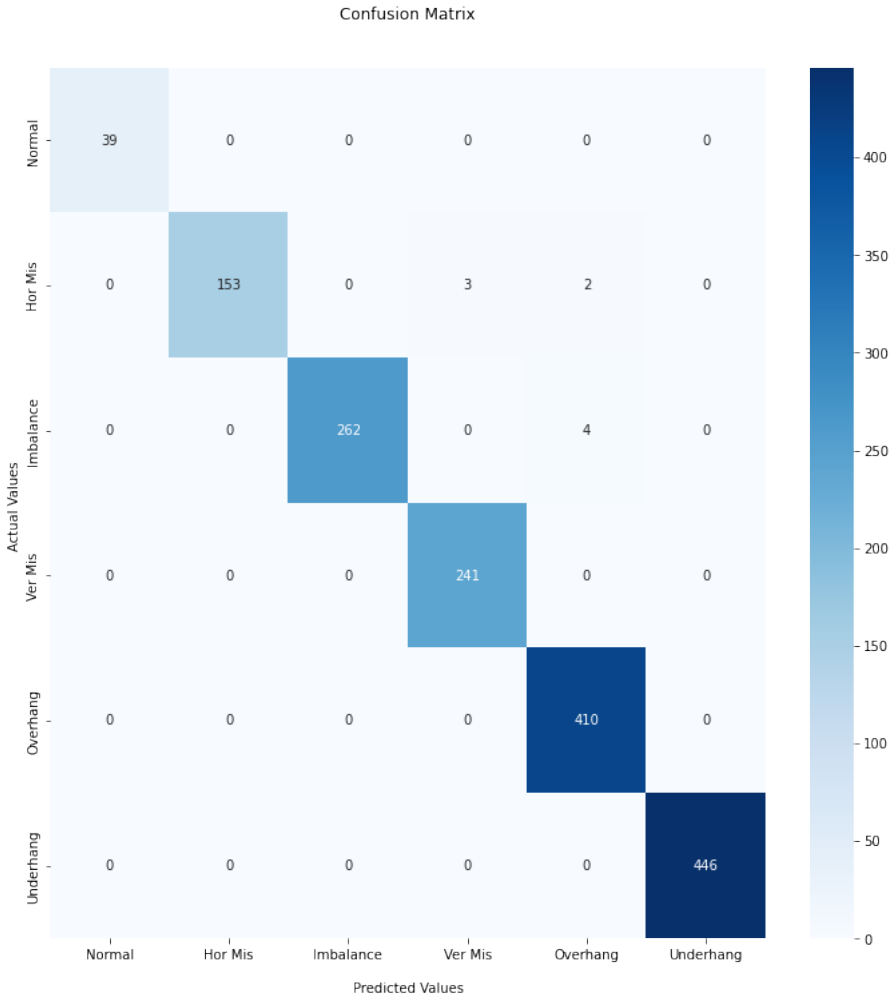


Figure 6.7: Confusion Matrix for Train Set

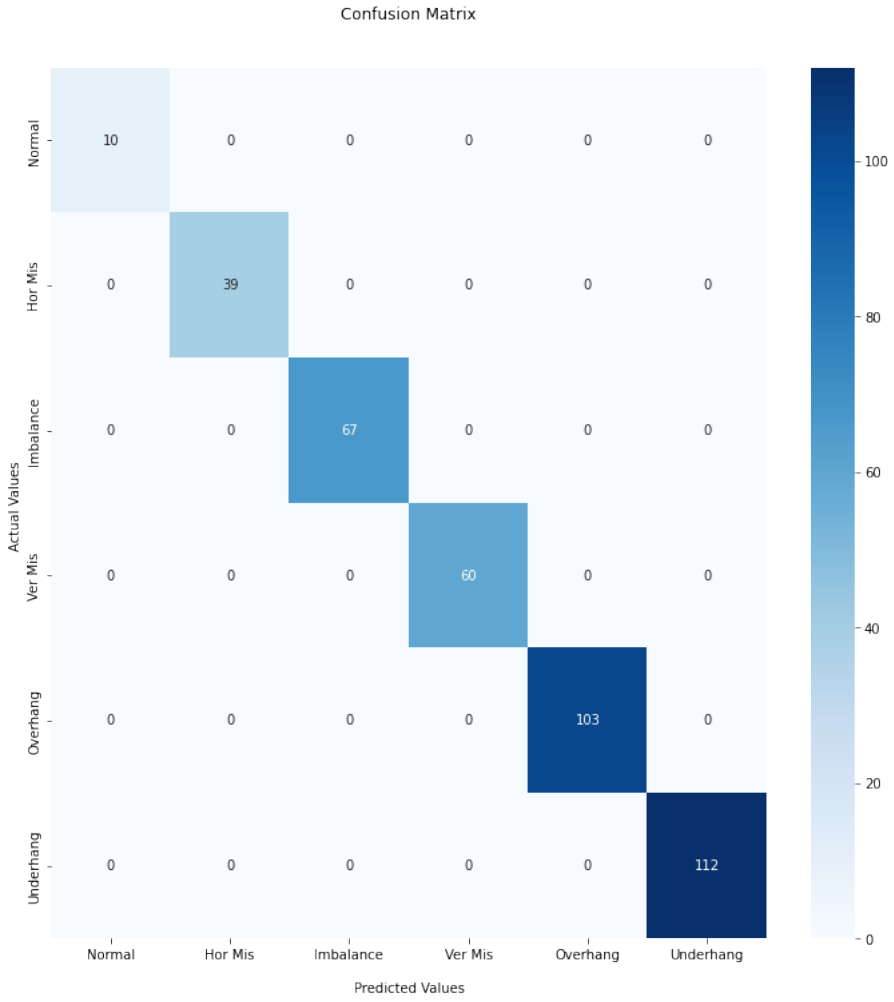


Figure 6.8: Confusion Matrix for Test Set

- The accuracy for class Ver Mis is: 100.0% = 241 / 241
- The accuracy for class Overhang is: 100.0% = 410 / 410
- The accuracy for class Underhang is: 100.0% = 446 / 446

Remarking that the two classes that did not reach an accuracy of 100% were Horizontal Misalignment and Imbalance.

What's more, as we saved up 200 models, we can easily retrieve their validation loss and accuracy. So that, refer to Figure 6.9 to see validation accuracy vs validation loss for all these models. It can be seen from this graph that the more accurate our model the lower loss we get (i.e. a clear evidence of a negative slope here). Thus, this shows that the selection of our loss: *Sparse Categorical Cross Entropy* was appropriate. In addition, referring to Figure 6.10, it can be seen the evolution of the validation accuracy through all these 200 different models, being this evolution for almost all the models above 95% accurate.

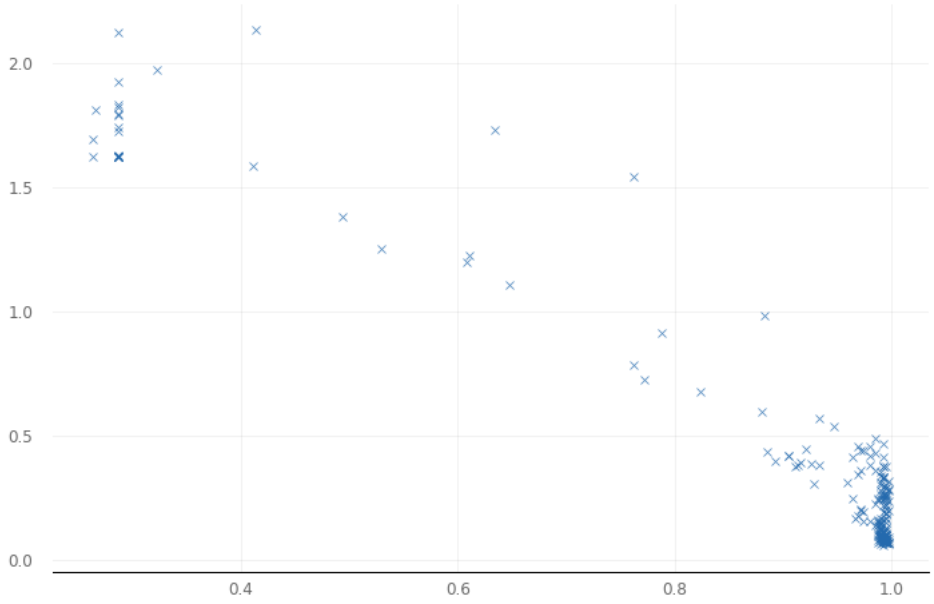


Figure 6.9: Validation Accuracy vs Loss

Lastly, it was calculated the kernel density estimator for the validation accuracy (Figure 6.11). From this graph, we can see how validation accuracy mainly lands somewhere above 95%, making this configuration a very reliable one.

When it comes to **Interpretability** using Grad-CAM techniques, the results found for this tuned-CNN are almost the same than for the non-tuned CNN. Please, refer to Figure 6.12 to have a visualization of an average heatmap for each class. Thus, we

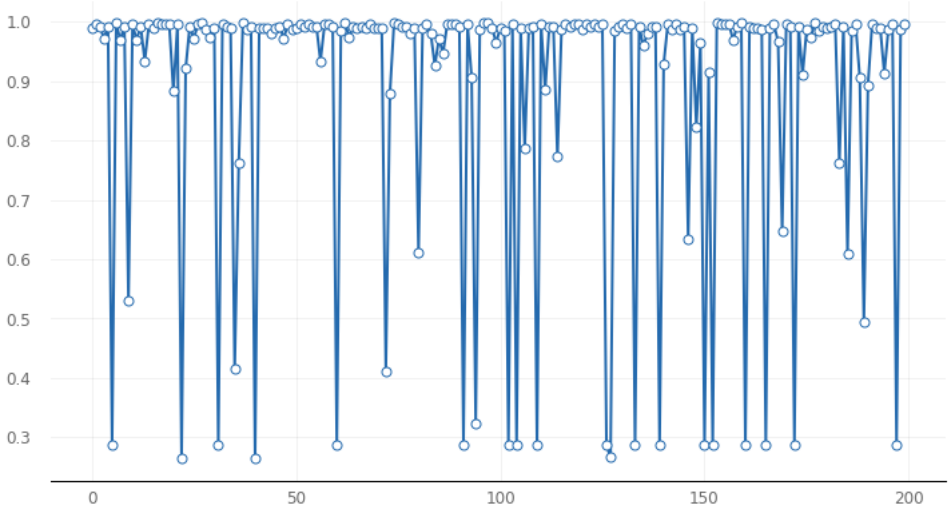


Figure 6.10: Validation Accuracy for 200 models

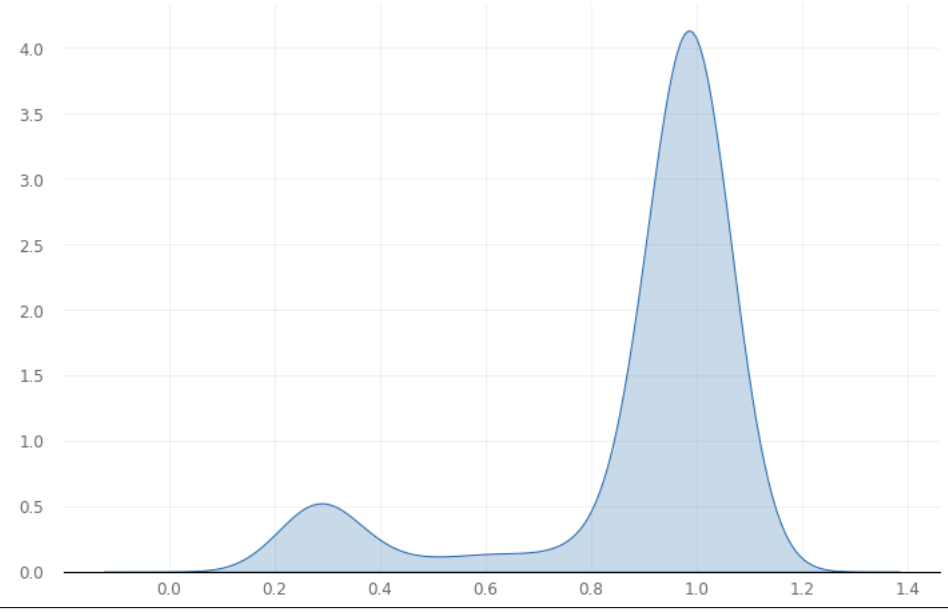


Figure 6.11: Kernel Density Estimator for Validation Accuracy

reinforce the reliability of the different parts of the vibration spectrum that the CNN takes into consideration in order to classify a single data point into each class.

However, the unique noticeable difference is that Underhang Class for this tuned-CNN is more clearly defined that for the previous tuned-CNN. Moreover, the bands that we can see from this, match with the results described in the next section 6.2.

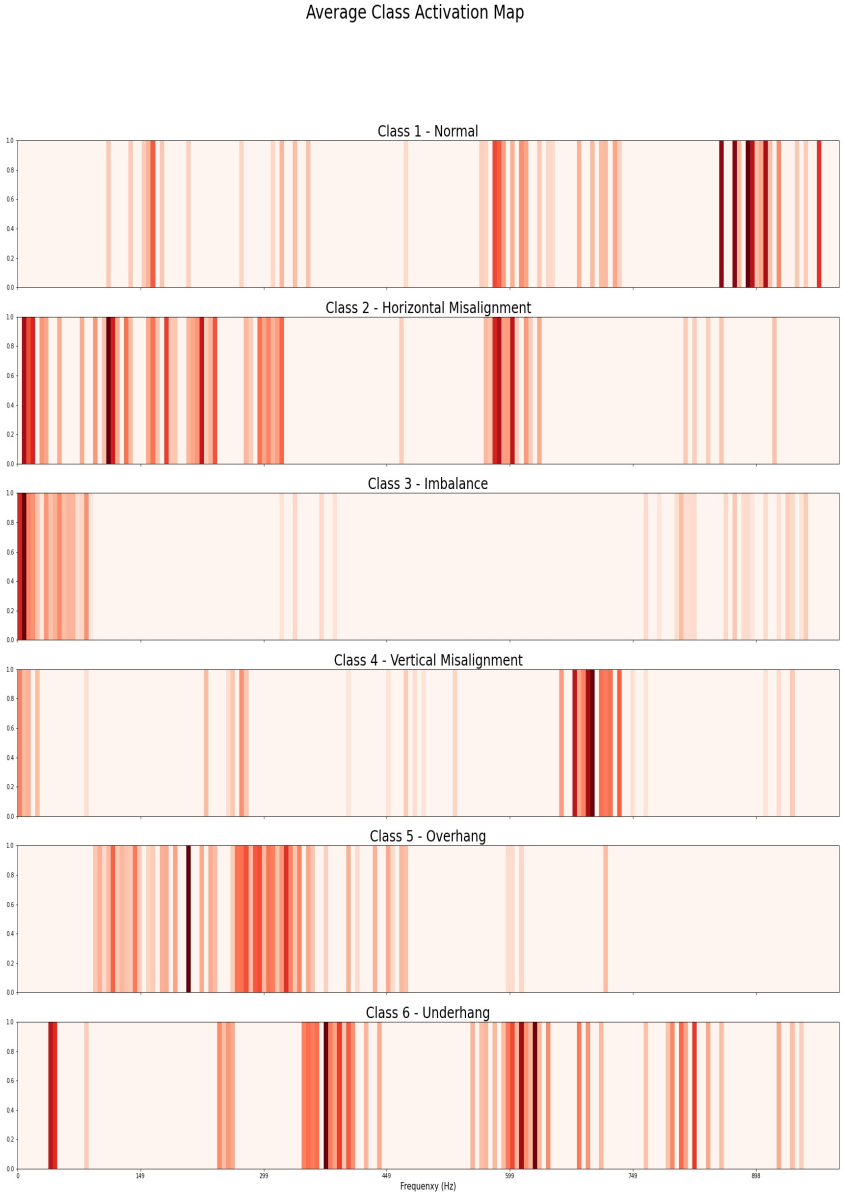


Figure 6.12: Average HeatMap for the tuned-CNN

6.2 Results for 2nd Experiment

In this experiment, the configuration of the CNN is exactly the same as in 5.2.1 , with the exception of the last Dense Layer having 4 neurons instead of 6.

As previously mentioned, the class activation heatmap for Underhang was not clearly defined and so, what we tried in this experiment is to isolate the different causes that this particular class has: cage fault, ball fault and outer race. The CNN also, was executed for 100 epochs and with the two same callbacks. The results in term of loss and accuracy are as follows:

- The minimum loss training found is: 0.051027 and was found in epoch number: 99
- The minimum loss test found is: 0.050591 and was found in epoch number: 99
- The maximum accuracy training found is: 100.0 % and was found in epoch number: 11
- The maximum accuracy test found is: 100.0 % and was found in epoch number: 9

The evolution of the loss function and accuracy along the way for these 100 epochs can be seen in Figure 6.13 and Figure 6.14. We can see exactly the same behaviour as with the previous experiment: the model converges to a high accuracy from epoch 15 and also, it does not present a lot of spikes making it reliable and consistent.

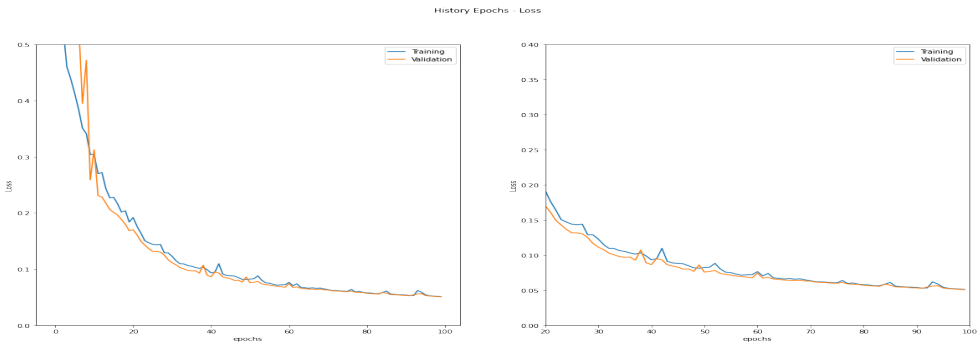


Figure 6.13: Loss Function - epochs from CNN

The confusion Matrix associate with this model can be seen in Figure 6.15. This Confusion Matrix reflects a 100% accurate model for training and test (i.e. not a single data point is misclassified).

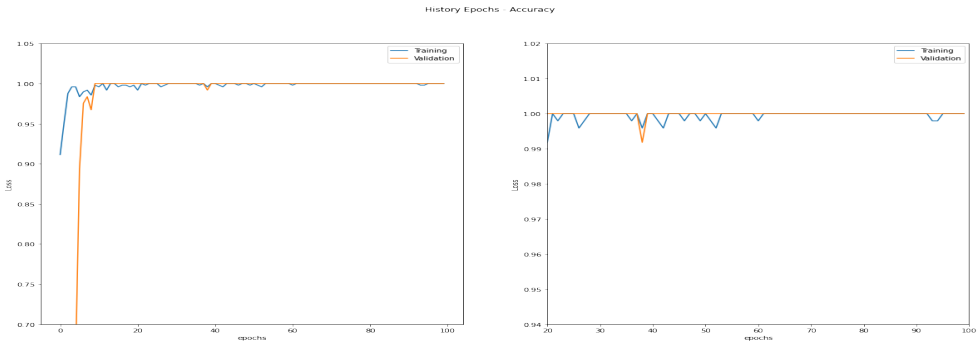


Figure 6.14: Accuracy - epochs from CNN

The gist of this experiment is to visualize the different zones that the CNN takes into account in order to classify a data point into the class Underhang. We followed the same strategy: averaging the heatmaps of each type class. Referring to Figure 6.16, we can see now a better allocation of the spectrum's zone of each class that trigger the decision. In particular, cage fault is allocated in vibration frequencies less than 50Hz, Outer race vibration frequencies are allocated betwee 250Hz and 380Hz and Ball Fault is allocated in two zones: 300-600Hz and 750-900Hz. Lastly, in Figure 6.17 it can be seen the average spectrum for each class in this experiment. As it happened before, the two previous figures match and the results make sense.

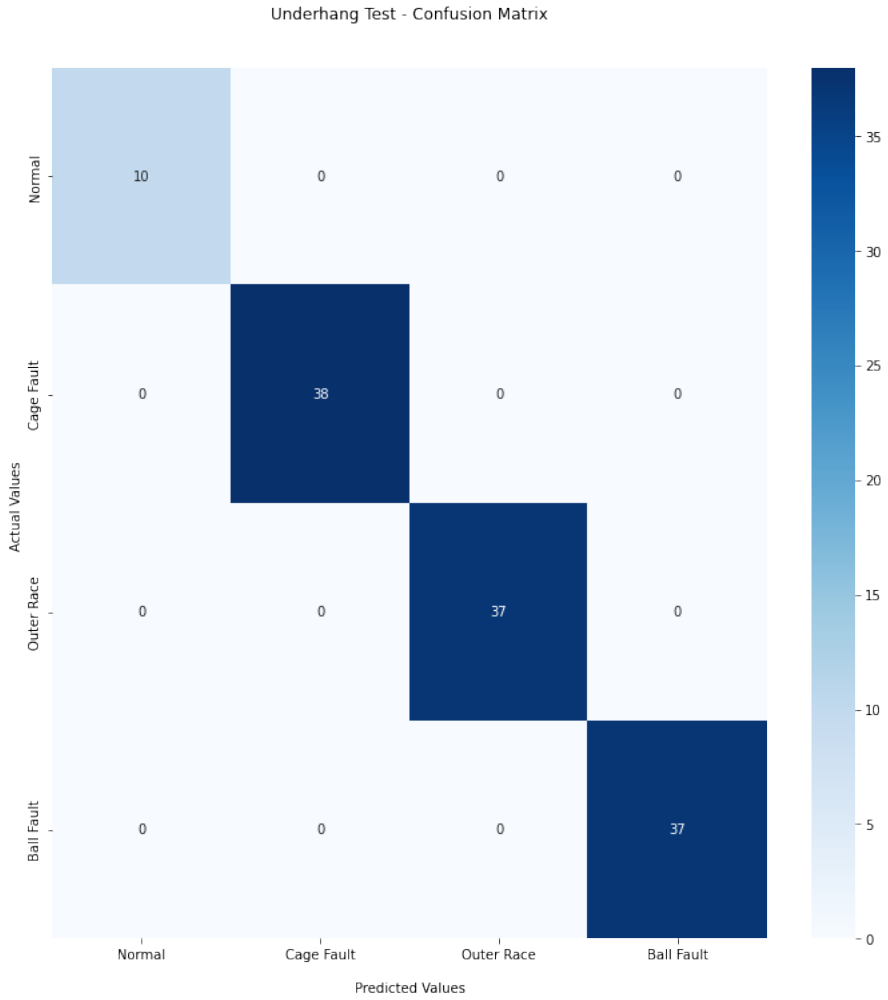


Figure 6.15: Confusion Matrix for 2nd experiment

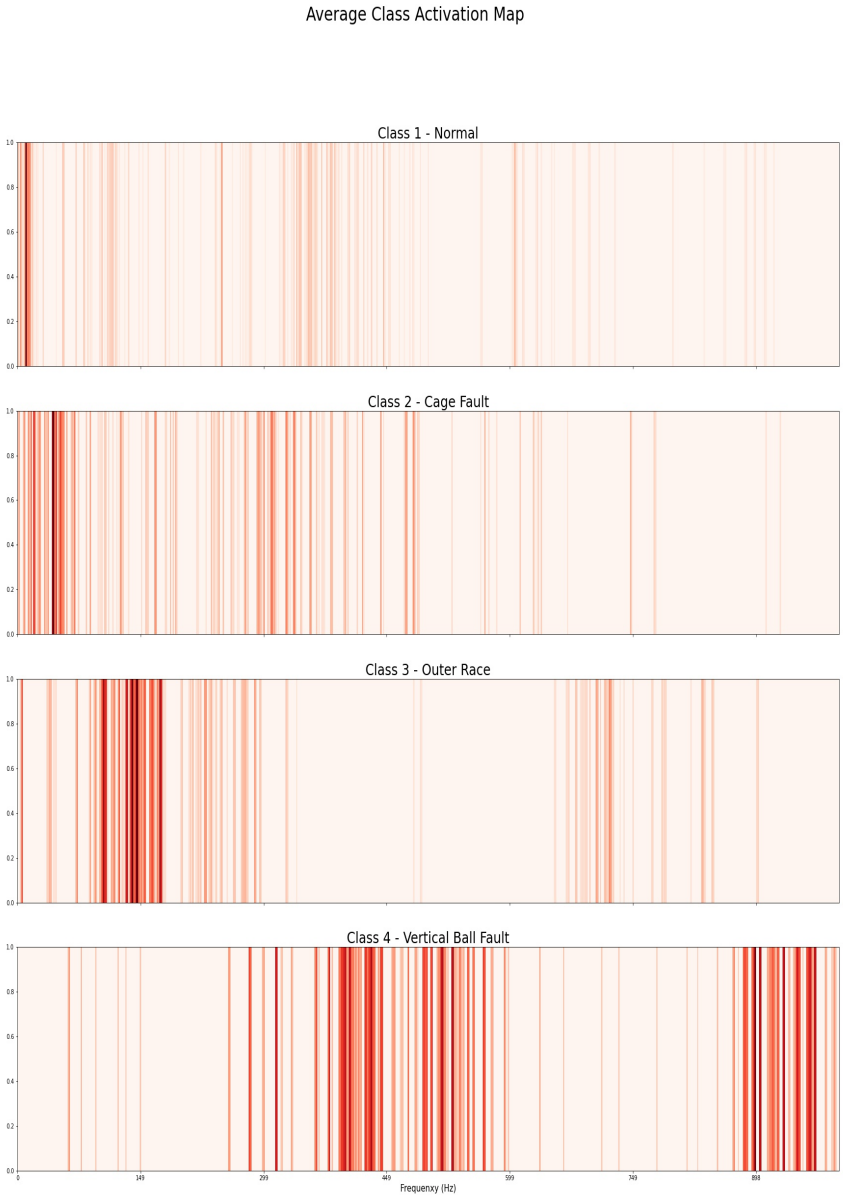


Figure 6.16: Average Heat Map for 2nd Experiment

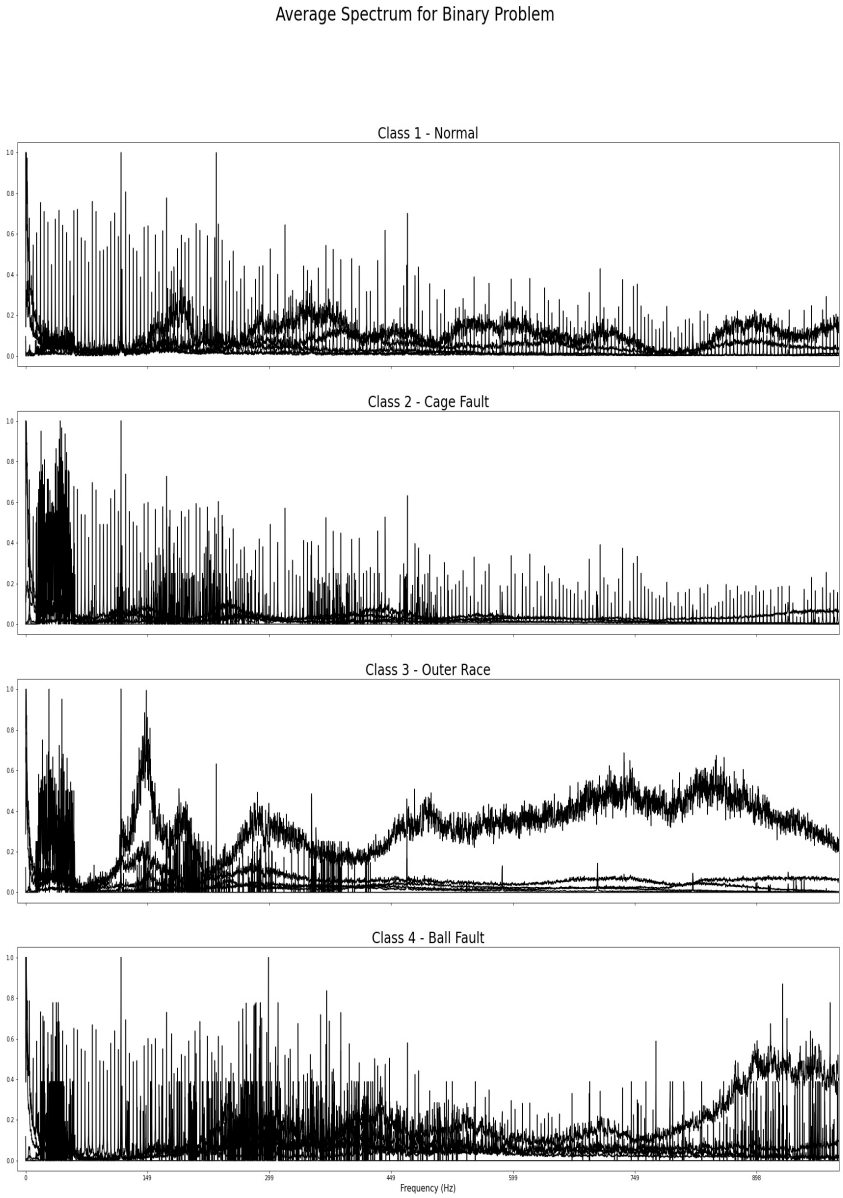


Figure 6.17: Average Vibration Spectrum for 2nd Experiment

Chapter 7

Future work

First of all, as we were not successful when it comes to simulating the proper behaviour of an induction motor, I would suggest that a possible future analysis of the problem is to build a simulation that hand with hand with a Deep learning model could predict the state of the motor. In order to build a simulation in SimuLink, it is worth studying the initial conditions of the motor and its metadata. This, is important, since the simulation should behave in a similar way as the motor that it is wanted to be simulated. Moreover, when the simulation was properly implemented, it would be also a data augmentation technique. So that, it could be generated data for different states of the motor and it could be dealt with the imbalance that usually these databases present. It should be highlighted that the imbalance usually appears since the motor almost always behaves in a healthy and operational state. Finally, as it could generate more data it would undeniably trigger an improvement in the precision of the Deep learning model involved, since the more data the model has the more information it can get from the properties that the data has under the hood. Therefore, it could be implemented a sufficient accurate deep learning model capable of detecting faulty states ahead of them happening.

The proposed CNN models, built with Deep learning with 1D-Convolutional Layers, have been shown to be capable of equalling the state-of-art which demonstrated an accuracy of 99.7% in the classification of failures using MAFAULDA database [Souza et al. 2021]. In addition, the results gave good interpretability of the frequency domain spectrum. An accuracy of 99.7% in the test means misclassifying an unique value, so that I would say that what is worth studying and applying in real life circumstances is the interpretability of the Deep learning model on the frequency domain. An interesting approach would be implementing this AI system with a real life motor and see if the results obtained for MAFAULDA database are similar to those that will be obtained by the new motor. In more detail, we could compare the different highlighted zones of the frequency domain spectrum for each state of the motors. If this comparison was reasonably similar, it could be said that the AI system was capable of extracting crucial information about each state of the motor and therefore, it could be capable also of

predicting at an early stage the possibility of a breakdown.

Another point is that our database was limited it would be worth studying this problem with a much bigger amount of data. This vast amount of data could be gained, as previously mentioned, using SimuLink as a data augmentation technique or merely using a more extensive database. With the use of a more s database, I would suggest that the zones highlighted by the Deep learning model will be even more highlighted. Therefore, it would be more defined the evidence of unhealthy frequencies in the frequency domain spectrum of an induction motor.

Furthermore, I would suggest implementing this whole analysis with different type of sensors. In this scenario, we focused on two accelerometers that were allocated in the underhang and overhang parts of the induction motor. Therefore, it would be worth studying, if for different kind of sensors we could get a similar and good results when it comes to the interpretability of this problem.

Moreover, if we implemented this proposed model in a motor but with different types of sensors inputs we could see if for different types of vibration signals the Deep learning model would be capable of having solid decisions. With solid decisions, I meant that for all types of sensors the classification would be the same.

Concerning the previous idea of having different sensors, it might be a good idea investigating if we could do transfer learning to our Deep learning model. A previous trained Deep learning model would share information about the motor to our CNN and we could see if the our model is capable of interpreting this features. With this approach, our model would be able to classify and give interpretability for a wide range of different sensors. This is a real practical implementation since the sensors allocated in each motor are likely to be different and so, they give different measures.

This transfer learning could come from more complex models. Therefore, I would suggest for future work to try implementing more complex models with more layers or a more complex structure in general. Building a more complex model might trigger an improvement in the performance in our problem and might make easier the transfer learning from one model to another.

Furthermore, a future approach would be implementing the model with different types of motors, not only with induction motors. Building a more complex model along with transfer learning from other models could be a solid idea in order to identify faulty states in a wide range of different motors. If the results for this possible approach were successful, we would have a Deep learning model capable of adapting to different sensors, motors and faulty states. Thus, the model would be able to catch at an early states different fault states for each particular motor.

With this idea, as companies usually have more than one type of motor, would be able to have a reliable software that monitor in real-time the functioning of all motors in their facilities. This monitoring, will make easier to catch breakdown and so, avoid financial and production losses.

Lastly, the model could be also modified in order to receive dynamic data (i.e. data

in real time). Then, it could be created a dashboard showing the frequency domain of each motor and the future technicians could oversee their functioning just looking at their monitors.

To end up with this thesis, I would like to give a personal view of the problem. In particular, this topic is fascinating since a lot of companies would take advantage of the Deep learning models. With them, the companies could save up of financial and production losses. However, this problem requires time in order to be studied and analysed properly and unfortunately, in a period of three months it was impossible for me. I would recommend getting a good GPU, storage and machine when it comes to analysing this problem since from my personal experience, only loading the data was used to take two hours. But as previously mentioned, the overall idea of this project is fascinating and I would like to highly encourage the reader in taking part of this project and continue it.

Bibliography

Alzghoul, Ahmad et al. [2021]. "On the Usefulness of Pre-processing Methods in Rotating Machines Faults Classification using Artificial Neural Network". In: *Journal of Applied and Computational Mechanics* 7.1, pp. 254–261.

Benbouzid, M El Hachemi [2000]. "A review of induction motors signature analysis as a medium for faults detection". In: *IEEE transactions on industrial electronics* 47.5, pp. 984–993.

Borovykh, Anastasia, Sander Bohte, and Cornelis W Oosterlee [2018]. "Dilated convolutional neural networks for time series forecasting". In: *Journal of Computational Finance, Forthcoming*.

Cinbis, Ramazan Gokberk, Jakob Verbeek, and Cordelia Schmid [2016]. "Weakly supervised object localization with multi-fold multiple instance learning". In: *IEEE transactions on pattern analysis and machine intelligence* 39.1, pp. 189–203.

Gan, Chuang et al. [2015]. "Devnet: A deep event network for multimedia event detection and evidence recounting". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2568–2577.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville [2016]. *Deep learning*. MIT press.

Heckbert, Paul [1995]. "Fourier transforms and the fast Fourier transform (FFT) algorithm". In: *Computer Graphics* 2, pp. 15–463.

Janssens, Olivier et al. [2016]. "Convolutional neural network based fault detection for rotating machinery". In: *Journal of Sound and Vibration* 377, pp. 331–345.

Jia, Feng, Yaguo Lei, Jing Lin, et al. [2016]. "Deep neural networks: A promising tool for fault characteristic mining and intelligent diagnosis of rotating machinery with massive data". In: *Mechanical systems and signal processing* 72, pp. 303–315.

Jia, Feng, Yaguo Lei, Na Lu, et al. [2018]. "Deep normalized convolutional neural network for imbalanced fault classification of machinery and its understanding via visualization". In: *Mechanical Systems and Signal Processing* 110, pp. 349–367.

Jogin, Manjunath et al. [2018]. "Feature extraction using convolution neural networks (CNN) and deep learning". In: *2018 3rd IEEE international conference on recent trends in electronics, information & communication technology (RTEICT)*. IEEE, pp. 2319–2323.

Layer weight regularizers [n.d.] Keras. URL: `https://keras.io/api/layers/regularizers/`.

Lin, Cheng-Jian, and Jyun-Yu Jhang [2021]. "Bearing fault diagnosis using a grad-cam-based convolutional neuro-fuzzy network". In: *Mathematics* 9.13, p. 1502.

Lin, Min, Qiang Chen, and Shuicheng Yan [2013]. "Network in network". In: *arXiv preprint arXiv:1312.4400*.

Mahendran, Aravindh, and Andrea Vedaldi [2016]. "Salient deconvolutional networks". In: *European conference on computer vision*. Springer, pp. 120–135.

Oquab, Maxime, Leon Bottou, et al. [2014]. "Learning and transferring mid-level image representations using convolutional neural networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1717–1724.

Oquab, Maxime, Léon Bottou, et al. [2015]. "Is object localization for free?-weakly-supervised learning with convolutional neural networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 685–694.

Ordóñez, Francisco Javier, and Daniel Roggen [2016]. "Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition". In: *Sensors* 16.1, p. 115.

Pestana-Viana, Denys et al. [2016]. "The influence of feature vector on the classification of mechanical faults using neural networks". In: *2016 IEEE 7th Latin American Symposium on Circuits & Systems (LASCAS)*. IEEE, pp. 115–118.

Phukon, Lakhya Jyoti, and Neelanjana Baruah [2015]. "A generalized MATLAB simulink model of a three phase induction motor". In: *International Journal of Innovative Research in Science, Engineering and Technology* 4.5, pp. 2926–2934.

Pinheiro, Pedro O, and Ronan Collobert [2015]. "From image-level to pixel-level labeling with convolutional networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1713–1721.

Qin, Lele, Naiwen Yu, and Donghui Zhao [2018]. "Applying the convolutional neural network deep learning technology to behavioural recognition in intelligent video". In: *Tehnički vjesnik* 25.2, pp. 528–535.

Rajagukguk, Rial A, Raden AA Ramadhan, and Hyun-Jin Lee [2020]. "A review on deep learning models for forecasting time series data of solar irradiance and photovoltaic power". In: *Energies* 13.24, p. 6623.

Ribeiro, Felipe M. L. [2022]. *MAFAULDA*. Online. URL: http://www02.smt.ufrj.br/~offshore/mfs/page_01.html.

Selvaraju, Ramprasaath R et al. [2017]. "Grad-cam: Visual explanations from deep networks via gradient-based localization". In: *Proceedings of the IEEE international conference on computer vision*, pp. 618–626.

Shi, KL et al. [1999]. "Modelling and simulation of the three-phase induction motor using Simulink". In: *International journal of electrical engineering education* 36.2, pp. 163–172.

Simonyan, Karen, Andrea Vedaldi, and Andrew Zisserman [2014]. "Visualising image classification models and saliency maps". In: *Deep Inside Convolutional Networks*.

Souza, Roberto M et al. [2021]. "Deep learning for diagnosis and classification of faults in industrial rotating machinery". In: *Computers & Industrial Engineering* 153, p. 107060.

Springenberg, Jost Tobias et al. [2014]. "Striving for simplicity: The all convolutional net". In: *arXiv preprint arXiv:1412.6806*.

Talos - Hyperparameter Optimization for TensorFlow, Keras and PyTorch [n.d.] GitHub. URL: https://github.com/autonomio/talos.

Terron-Santiago, Carla et al. [2021]. "A Review of Techniques Used for Induction Machine Fault Modelling". In: *Sensors* 21.14, p. 4855.

Wibawa, Aji Prasetya et al. [2022]. "Time-series analysis with smoothed Convolutional Neural Network". In: *Journal of big Data* 9.1, pp. 1–18.

Zeiler, Matthew D, and Rob Fergus [2014]. "Visualizing and understanding convolutional networks". In: *European conference on computer vision*. Springer, pp. 818–833.

Zhang, Wei, Chuanhao Li, et al. [2018]. "A deep convolutional neural network with new training methods for bearing fault diagnosis under noisy environment and different working load". In: *Mechanical Systems and Signal Processing* 100, pp. 439–453.

Zhang, Wei, Xiang Li, et al. [2020]. "Machinery fault diagnosis with imbalanced data using deep generative adversarial networks". In: *Measurement* 152, p. 107377.

Zhang, Wei, Gaoliang Peng, et al. [2017]. "A new deep learning model for fault diagnosis with good anti-noise and domain adaptation ability on raw vibration signals". In: *Sensors* 17.2, p. 425.

Zhou, Bolei et al. [2016]. "Learning deep features for discriminative localization". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2921–2929.