Lifetime Based Health Indicator for Bearings using Convolitional Neural Networks

ARILD BERGESEN HUSEBØ

SUPERVISORS
Van Khang Huynh
Jagath Sri Lal Senanayaka

University of Agder, 2019
Faculty of Engineering and Science
Department of Engineering Sciences

Preface

In august 2018 I started working on a university project which resulted in a publication and an oral presentation for WEMDCD 2019 in Athens, Greece. This was the beginning of my journey into applied machine learning and condition based monitoring, and ever since the project was submitted I have been exploring the possibilities of combining these two disciplines. The work presented in my M.Sc. thesis is one of the most novel ideas that has came to mind after working on the previous project, thus it became what I wanted to spend the previous months pursuing. I would like to thank Van Hyunh Khang who has been my supervisor both now and during the previous project, always pointing me in the right direction. I would also like to thank Jagath Sri Lal Senanayaka for supervising my M.Sc. thesis project. Lastly, I would like to thank my university faculty for making everything possible.

Arild Bergesen Husebø

Grimstad, May 2019

Abstract

Out of all the components in rotating electrical machinery, bearings have the highest failure rate. Bearing degradation is a seemingly random process which is hard to both model and predict. Countless of condition based methods and algorithms have been proposed in order to accurately diagnose incipient faults and estimate the remaining useful lifetime of bearings. These methods are often complex and hard to implement. In this thesis, a data-driven method of estimating a linear lifetime based health indicator (HI) using convolutional neural networks (CNNs) is proposed. The idea behind the method is to train a CNN model to recognize the shapes and distributions of vibration data in order to predict a HI with minimal pre-processing. Two models are presented: A CNN that takes time-series vibration data as input and a CNN that takes vibration frequency spectrum data as input. Finally, HIs are predicted on unique datasets and their respective remaining useful lifetimes (RULs) are estimated as part of the model validation process. The results show that the models are able to recognize relevant fault features to a certain degree. However, accurate predictions have proven difficult in many cases.

Individual/group Mandatory Declaration

The individual student or group of students is responsible for the use of legal tools, guidelines for using these and rules on source usage. The statement will make the students aware of their responsibilities and the consequences of cheating. Missing statement does not release students from their responsibility.

1.	I/We hereby declare that my/our thesis is my/our own work and that I/We have not used any other sources or have received any other help than mentioned in the thesis.	☒
2.	I/we further declare that this thesis: - has not been used for another exam at another department/university/university college in Norway or abroad; - does not refer to the work of others without it being stated; - does not refer to own previous work without it being stated; - have all the references given in the literature list; - is not a copy, duplicate or copy of another's work or manuscript.	☒
3.	I/we am/are aware that violation of the above is regarded as cheating and may result in cancellation of exams and exclusion from universities and colleges in Norway, see Universitets- og høgskoleloven §§4-7 og 4-8 og Forskrift om eksamen §§ 31.	☒
4.	I/we am/are aware that all submitted theses may be checked for plagiarism.	☒
5.	I/we am/are aware that the University of Agder will deal with all cases where there is suspicion of cheating according to the university's guidelines for dealing with cases of cheating.	☒
6.	I/we have incorporated the rules and guidelines in the use of sources and references on the library's web pages.	☒

Publishing Agreement

Authorization for electronic publishing of the thesis.

Author(s) have copyrights of the thesis. This means, among other things, the exclusive right to make the work available to the general public (Åndsverkloven. §2).

All theses that fulfill the criteria will be registered and published in Brage Aura and on Ui A's web pages with author's approval.

Theses that are not public or are confidential will not be published.

I hereby give the University of Agder a free right to

make the task available for electronic publishing: ☒ JA ☐ NEI

Is the thesis confidential? ☐ JA ☒ NEI

(confidential agreement must be completed)

- If yes:

Can the thesis be published when the confidentiality period is over? ☐ JA ☐ NEI

Is the task except for public disclosure? ☐ JA ☒ NEI

(contains confidential information. see Offl. §13/Fvl. §13)

Contents

List of Figures

List of Tables

1 Introduction

Components of rotating machinery, no matter the application, are prone to degradation and eventually failure. Bearings are necessary components of almost all rotating machinery, and at the same time the component that most often fails. Data on from IEEE shows that more than 40% of faults occuring in induction motors are bearing faults [1, p. 13]. Figure 1 gives an overview of these fault distributions. However, for large machines, this percentage can reach as high as 90% [2].

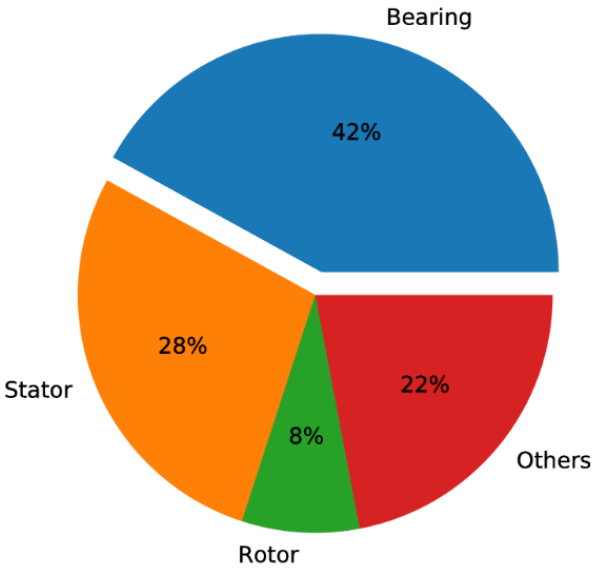

Figure 1: Induction motor fault distribution [1]

It is desired to detect these incipient failures before they can fully develop, potentially disabling a system for a period of time. This period of time can be drastically reduced if one can proactively predict when the failure will happen. Traditional methods of detecting faults involves manual maintenance and making inspections of the machines by hand. This is both an expensive and time-intensive approach, although most bearing faults can be avoided by taking preventive measures like substitution and lubrication periodically [3]. However, more and more automated methods have emerged utilizing either online or offline tools with underlying diagnostic algorithms to detect incipient failures. These methods go under the category of condition based monitoring. Condition based monitoring can be performed either model driven or data driven. Model driven monitoring is based on comparing observations of a system to a fixed mathematical model. On the other hand, data driven monitoring is when the model is shaped based on the observations of the system. Some monitoring methods also combine these two methods into a hybrid variant [4].

Recent state-of-the-art work into diagnosis and prognosis of bearings has shown several new possible approaches [3]. The following are some of the recent findings in the field: Acoustics signals have proven a possible candidate to detect bearing faults, even with data recorded from a mobile phone [5]. The external stray flux around a motor has been used as a diagnostic method to detect general roughness in a bearing [6]. However, due to bearing fault signatures being significantly weaker in electromagnetic signals, some authors propose motor current signature analysis (MCSA) approaches to study the systems under steady-state operating conditions [7]. With varying air gap eccentricity and unbalanced magnetic pull (UMP) due to bearing faults, the fault features will testify themselves in the air gap power, thus also the motor currents.

Most of the mentioned research mainly focuses on being able to detect the fault signatures and do not present implementations of full diagnostic or prognostic systems. This means that a lot of the research mainly focuses on developing advanced algorithms for fault detection only. However, some papers present

full diagnostic and prognostic methodologies: Vibrational data can be used to calculate remaining useful lifetime (RUL) based on the time of critical current discharge events [8], due to the phenomenon of Electrical Discharge Machining (EDM) when working with high-frequency pulse width modulation (PWM) converters, that induce AC currents in the shaft that accelerates the bearing degradation [9]. Also, reliable health indicators (HI) can be estimated by employing the discrete wavelet packet transform (DWPT) to decompose vibration signals into sub-bands, and extracting the HI from each sub-band. Then, the HIs showing the best trends can be accumulated to construct the overall HI giving the best trend throughout the bearing lifetime [10].

Over the recent years, there has been a lot of research into applying machine learning methodologies to perform diagnosis and prognosis on electrical machines. Several methods using artificial neural networks have been explored by researches, some of them being the following: Current signature analysis has been used in a combination with a deep 1D-convolutional network to diagnose faulty motor states [11]. Sparse deep stacking networks have been trained on vibration data mapped to a feature space in order to overcome the overfitting risk of complex deep networks [12]. For induction motor fault diagnosis, a convolutional discrimative feature learning methodology has been applied to learn the shapes of relevant fault features [13].

Simplicity is often better than complexity. Much of the current research into machine learning approaches focuses heavily on training deep neural network structures or combining advanced feature extraction methods with machine learning algorithms. The previous work *Convolutional Neural Networks for bearing fault diagnosis* (Husebø, Huynh & Pawlus, 2019) focused on diagnosing fault type and severity based on classification using convolutional neural networks with time series and continuous wavelet transform (CWT) spectrogram vibration data [14]. Run-to-failure datasets were split into healthy, degraded and faulty condition classes of their respective fault types, eg. healthy, degraded inner race ring, faulty outer race ring. The work showed good proof-of-concept, but still suffered from a couple of practical drawbacks. First and foremost, the convolutional neural network models were validated on data from the same datasets used for training the models. Even though the data was unique, due to the k-folds cross-validation method applied, the validation data still represents the same bearings running under the same conditions as the training data. Secondly, the data was labeled by hand based on the RMS-plots of the datasets. As will be discussed later in this thesis, the RMS-plots of a run-to-failure dataset does not always give a good representation of the actual degradation process.

The research presented in this thesis seeks to address these issues by training convolutional neural networks to estimate a floating linear HI that bases itself on the RUL of the system. First, a convolutional neural network is trained on temporal vibration data, then another convolutional neural network is trained on complex frequency spectrum data. Although the bearings tested in this thesis are rolling-element type ball bearings, the methodology does apply for other types of rolling-element bearings. The thesis is structured as follows: Section 2 tries to establish the theory behind bearing degradation processes. Section 3 contains the proposed method. Subsection 3.1 describes the data prework methodology. Subsection 3.2 explains the convolutional network theory, methodology and the final structures. Section 4 describes the experimental setup and data collection as well as final results and discussion regarding the outcome of network training and predictions.

2 Bearing Degradation

The degradation process of bearings is complex and there is no accurate model describing it using all the relevant parameters. Things to consider are operational factors such as speed, loading, materials and environment. The operating speed of bearings is directly linked to the amount of load cycles going into degrading the bearings. The load increases the magnitude of each load cycle, thus increasing the degradation rate. An imbalanced load will greatly affect the degradation rate as certain resonance frequencies within the shaft will occur. The material of bearing race ways and roller elements is also a factor to consider. Bearings can be made up of several different materials, including different steel alloys or ceramics, each with their own characteristics [15].

2.1 Fault features

Analytically, certain fault feature frequencies of bearings can be associated with the bearing geometry. A degradation defect in a roller element for instance, will correspond to the frequency of the individual spin velocity of that roller element as shown in Equation 1

$$BSF = \frac{d_p}{2d_b} f_s (1 - (\frac{d_b}{d_p})^2 (\cos\theta)^2) \tag{1}$$

where BSF is the ball spin frequency, d_p is the ball path diameter, d_b is the ball diameter, f_s is the rotational shaft frequency and θ is the angle of ball contact. A defect in the outer or inner race ways will correspond to the frequency of the roller elements passing over the location of said defect, as shown in Equations 2 and 3

$$BPFO = \frac{N_b}{2} f_s (1 - \frac{d_b}{d_p} \cos\theta) \tag{2}$$

$$BPFI = \frac{N_b}{2} f_s (1 + \frac{d_b}{d_p} \cos\theta) \tag{3}$$

where $BPFO$ is the outer race way ball pass frequency and $BPFI$ is the inner race way ball pass frequency. Lastly, the frequencies of a defect in the roller element cage, or fundamental train, will correspond to the rotation of the fundamental bearing rotation, or shaft speed, and its harmonics. The fundamental train frequency (FTF) is shown in Equation 4.

$$FTF = \frac{f_s}{2} (1 - \frac{d_b}{d_p} \cos\theta) \tag{4}$$

Figure 2 shows the basic geometry of a rolling element bearing consisting of seven roller elements. The symbols of the Equations 1 to 4 relate to the symbols of the figure.

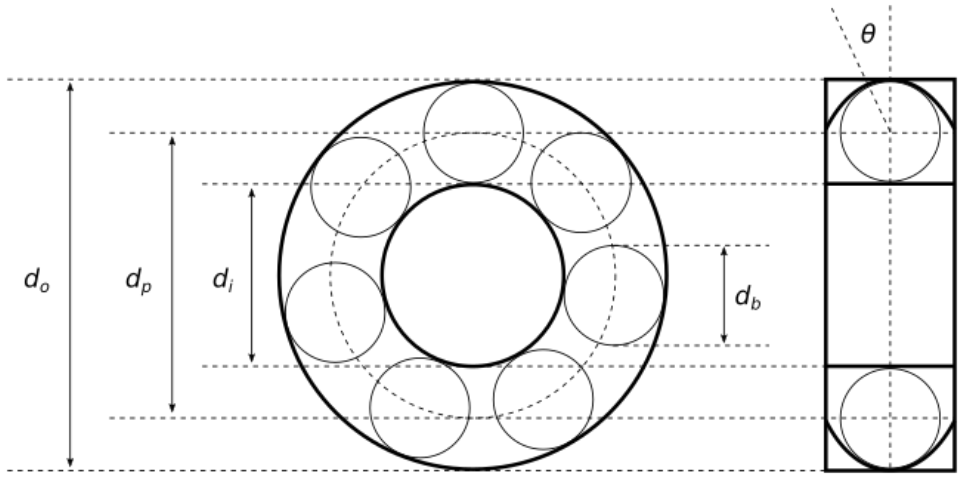

Figure 2: Bearing geometry

As mentioned, the equations are analytical and do not take into account other factors affecting the spectrum of a bearing. Figure 3 shows an accelerometer vibration signal and its corresponding frequency spectrum acquired using the Fast Fourier Transform (FFT). The signal is sampled from a bearing with a degradation artifact operating at 1800 rpm with a 4000 N load. Making out these analytical feature frequencies just by looking at the spectrum is hard. Other noises, harmonics and sidebands often show larger magnitudes than the actual frequencies of interest.

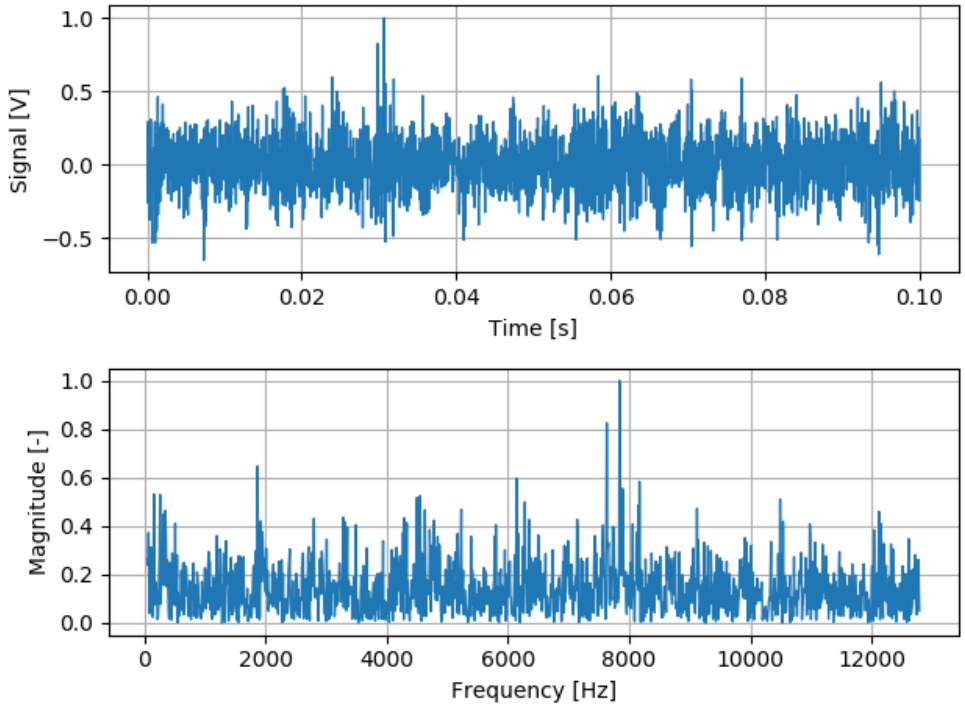

Figure 3: Time series signal and spectrum of degraded bearing vibration

2.2 Degradation Modelling

The actual bearing degradation process itself is hard to model. If a bearing defect is considered as a crack, Paris' law can be attributed to the process [16]. Paris' law was postulated in the 1960's by P. Paris and F. Erdogan and states that the range of stress intensity factor, ΔK, might characterise sub-critical crack growth under fatigue loading in the same way that it characterises critical fracture. This made it possible to make a quantitative prediction of residual life for a crack of a certain size [17]. The best known form of the law is shown in Equation 5.

$$\frac{da}{dN} = C\Delta K^m \tag{5}$$

where $\frac{da}{dN}$ is the crack growth rate, thus crack growth da per load cycle dN. C and m are material-dependent constants and ΔK is the range of stress intensity factor during the fatigue cycle.

Assuming the elapsed load cycles proportional to the current lifetime of a bearing under constant speed and load conditions, integrating the equation with respect to dN will give us an exponential function of the crack length a at a certain point in time. However, it is not as simple as modeling bearing degradation as an exponential model with respect to just a couple of operational and material dependent parameters. Bearing degradation is considered a stochastic process, meaning that there is a certain randomness to the propagation of the crack. The fact that the spectrum of Figure 3 is not purely made up of the fault feature frequencies is good evidence of this. Therefore, it is desired to model degradation using both deterministic and stochastic parts, where the deterministic part represents a constant physical phenomenon which is common across all systems, and the stochastic part that captures the specific variations of an individual system [18]. Thus, due to the stochastic nature of the degradation processes, it has always been hard to do accurate condition based prognosis of degradation processes.

2.3 Monotonicity

Another important aspect of degradation processes is their monotonic nature. Bearing degradation is an irreversible process, thus the function of degradation versus time will always either stay constant or increase. The Spearman's rank correlation coefficient can be used to determine the monotonicity of such a function. It is defined as the Pearson (linear) correlation coefficient between the rank variables [19]:

$$r_s = p_{rg_X, rg_Y} \tag{6}$$

where r_s is the Spearman correlation coefficient and p_{rg_X, rg_Y} is the Pearson correlation coefficient applied to the rank variables. Using the definition of the Pearson correlation, this expression is expanded to Equation 7:

$$r_s = \frac{cov(rg_X, rg_Y)}{\sigma_{rg_X}\sigma_{rg_Y}} \tag{7}$$

where $cov(rg_X, rg_Y)$ is the covariance of the raw scores X_i, Y_i converted to rank variables and $\sigma_{rg_X}, \sigma_{rg_Y}$ are the standard deviations of the rank variables. Both the Pearson and Spearman correlation coefficients will be used to value the final predictions made by the convolutional neural network models.

3 Method

3.1 Data Preparation

Training a neural network usually requires a lot of unique data. In addition to training data, one must supply data for validation the model after it has been trained. The validation data can not be identical to the data that was used during training, and should optimally stem from entirely different experiments. This thesis proposes a methodology for training two different convolutional neural network models to recognize a linear health indicator based on the remaining lifetime of bearings. The data should therefore come from run-to-failure experiments. One of the models is trained on time-series data, while the other one is trained on complex frequency spectrum data. The proposed method flowchart is shown in Figure 4.

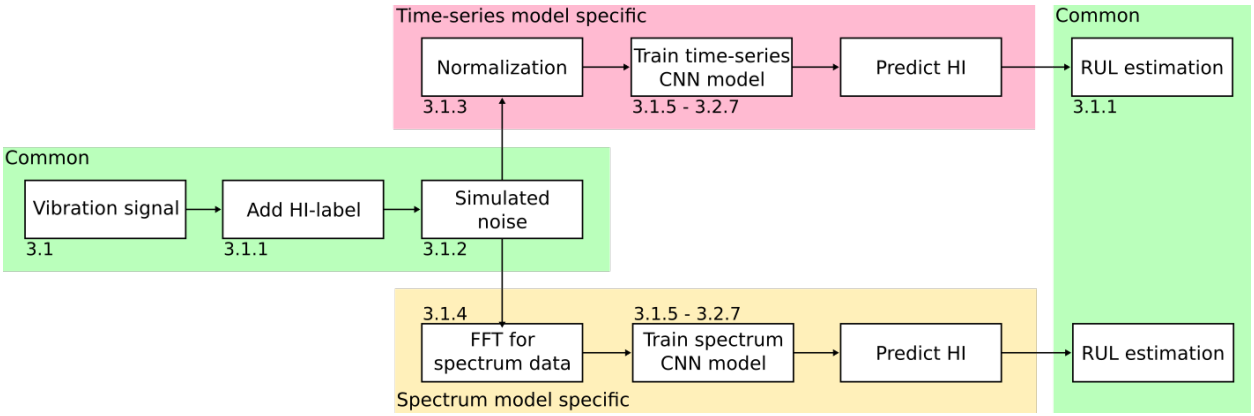

Figure 4: Method flowchart

3.1.1 Health indicator labels

The convolutional neural network model can be thought of as a non-linear regression model used to predict a percentage of total degradation value. This value, called the health indicator (HI), specifies how far the bearing has progressed throughout its lifetime. This means that at 50%, the bearing has lived through half of its total lifetime. Thus, the HI will be a linear function of time, from 0% at the beginning of operation, to 100% at the time of total failure. Labeling each signal with such a linear HI makes it easier to estimate the predicted time of total bearing failure. If the HI measured the defect size of an actual physical degradation, for instance using an exponential model as mentioned earlier, one would only be able to start estimating total lifetime once the bearing neared total failure and the HI gradient start increasing. One could argue that the gradients could be computed in a logarithmic scale and transformed back, but as predictions always will have a certain error, one would not be able to differentiate early HI growth from prediction errors. An example of the phenomena is shown in Figure 5. The early HI errors are amplified in the logarithmic plot, thus making it impossible to estimate any gradient.

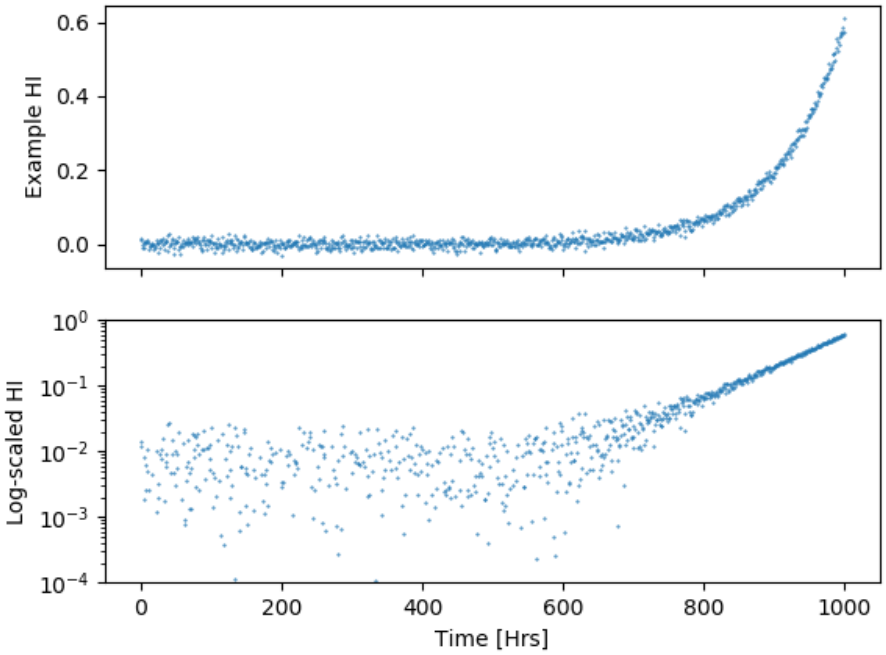

Figure 5: Example of exponential HI predictions.

A run-to-failure dataset consists of files that each make up samples of vibration signals. Using the file index sorted chronologically with respect to time and knowing the length of the dataset, the HI is calculated as shown in Equation 8.

$$HI_s = \frac{n_s}{N} \tag{8}$$

where HI_s is the health indicator at a specific sampling file s, n_s is the file number of the file and N is the length, or number of files, of the dataset. This way, the remaining useful lifetime (RUL) can be calculated using Equation 9:

$$RUL = \frac{(1 - HI_s)t_s}{HI_s} \tag{9}$$

where t_s is the current lifetime of the bearing at specific sampling file s. In order to create a smooth HI and RUL trend, a moving average (MA) with a window size of 100 is applied to the predictions.

Another method for estimating the RUL based on the predicted HIs is proposed and tested; a linear least squares approximation of all the predicted HIs up to the current point in time t_s. A general representation of a linear function with no constant part, ie. $f(t) = v_d t$ is used, where the speed of degradation v_d is approximated. A vector x is computed to minimize the Euclidean 2-norm $||b - ax||^2$, where a is the matrix of coefficients and b is the ordinate values to be approximated [20]. Let the prediction number of a dataset at time t_s be n_{t_s}. Then matrix a will look like

$$\begin{bmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 1 \\ \vdots & \vdots \\ n_{t_s} & 1 \end{bmatrix}$$

b is a vector containing all the predicted HIs up to time t_s.

3.1.2 Simulated noise

The experiments were conducted in controlled environments. In industrial applications, this will not be the case and there will to a certain degree be some environmental noises. In order to test the robustness of the convolutional neural network models, Gaussian noise is added to the signals. The noise scale is set to 20% of the signal standard deviation during training and 10% during validation. Applied noise during training is a helpful technique to help the network with generalization, and can also be applied internally in the neural activations themselves if needed [21].

3.1.3 Normalization

We are not interested in training the neural network to recognize overall signal magnitude. Assuming vibration magnitude increases proportionally with applied load, each signal is normalized in order to make the model load-independent. This will also generalize in cases were accelerometers are not calibrated, ie. have varying output gains. Recent research also shows that normalization of input data accelerates the network training process as the value ranges are kept constrained [22]. Each sample is divided by the infinity normal of the signal as defined in Equation 10.

$$z_i = \frac{x_i}{\|x\|_\infty} = \frac{x_i}{\max(|x_1|, ..., |x_n|)} \tag{10}$$

where z_i is the normalized sample i of signal x and n is the length of the signal. The normalized signals will lie in the range of -1 to 1.

3.1.4 Fast Fourier Transform for spectrum data

One of the most common methods of transforming time series data into frequency domain data is using the Fourier transform. The spectrum of $F(x)$ for a continuous function $f(t)$ is given by Equation 11

$$F(x) = \int_{-\infty}^{\infty} f(t) e^{-jtx} dt \tag{11}$$

where j is the imaginary unit. This is know as the continuous Fourier transform (CFT). Adapting the continuous Fourier transform to work with discrete data gives us Equation 12, which is known as the discrete Fourier transform (DFT)

$$X_k = \sum_{n=0}^{N-1} a_m e^{-2\pi j \frac{nk}{M}} \tag{12}$$

where a is a data sequence of length N.

However, as this transform relies on transforming each discrete data point of a time series signal for each discrete frequency, it quickly becomes an expensive operation. The total amount of computations required for a signal of length n would equal $2N^2$. Therefore, Fast Fourier transforms (FFTs) are applied to make approximations of the DFT. There are several FFT algorithms, but this work utilizes the Cooley-Tukey algorithm [23]. This algorithm reduces the amount of computations to $2N \log_2 N$ by doing a decimation in time [24].

The second convolutional neural network model is trained on complex positive-sided absolute-valued spectrum data estimated by using the FFT on time-series data. This gives both frequency magnitude information (real part), as well as phase information (imaginary part). The spectrum is not estimated on normalized time-series data, but rather unitarily transformed after by scaling the spectrum by $\frac{1}{\sqrt{n}}$ where n is the amount of frequency sample points.

3.1.5 Data training feed

During training of the network, each file of the training datasets is given a unique ID. All the IDs are then shuffled. Training neural network models on shuffled data has proven to be more efficient [25]. The signals corresponding to their respective IDs are dynamically loaded into memory and fed in batches of size 64 during each epoch of training. Figure 6 illustrates the process.

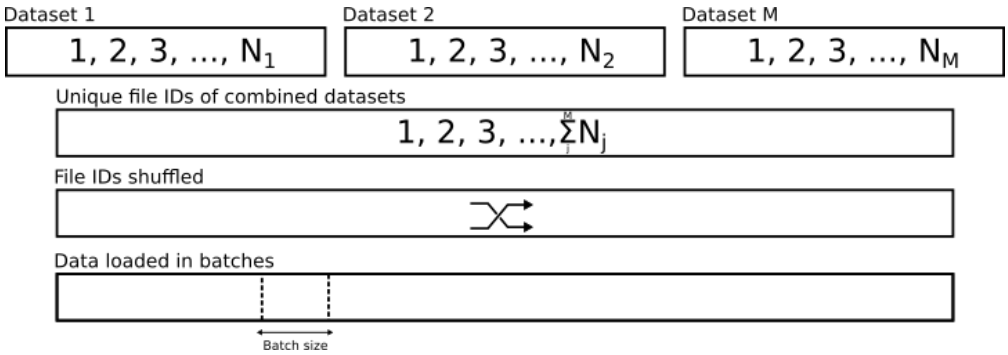

Figure 6: Data feeding process

During the final validation predictions, the CNN models are fed in batches of size 1. The actual HI is stored in pairs with the predicted HI of each signal.

3.2 Convolutional Neural Networks

The convolutional neural network is an extension of the artificial neural network (ANN). Artificial neural networks is an idea developed in the mid 1900s to replicate the processes happening within our brain [26]. They are made up of layers of neurons activated when a signal is received, analogous to the behavior of the brain. Layers of neurons are connected by weights, in turn determining which signals going into a neuron matter more. This is similar to how the axons in brain cells lead electrical impulses to the synaptic receptors of another brain cell.

A neuron of an ANN can defined as in Equation 13

$$y = \sigma(w \cdot x + b) \tag{13}$$

where y is the output signal of that neuron, σ is the activation function, w is the vector of weights of the connected neurons of the previous neural layer, x is the vector of input signals from the connected neurons of the previous neural layer and b is an activation bias. Note that $w \cdot x$ means the dot-product between the vectors, resulting in a scalar. This essentially means that the activation of a given neuron depends on the sum of element-wise multiplication of the signals and their respective weights. Figure 7 visually illustrates the concept: The neuron in question (green) recieves a signal from each of the three connected neurons. Each of these signals is multiplied by their respective weights, and finally the bias is added. The activation function then takes this value as its input and outputs a new value, which is the output of that neuron.

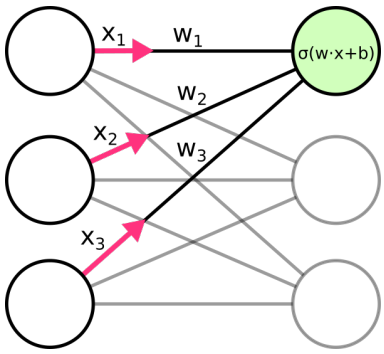

Figure 7: Concept of artifical neurons

Neural networks can be trained either supervised or unsupervised, depending on the task. Supervised learning is similar to regression problems, where it is trained to replicate a set of training data, called labels. This is commonly done by training the network on one pair of input and output data, but complex neural network models can have multiple inputs and outputs. Examples of supervised learning techniques are shown in traditional neural network classifiers, support vector machines (SVM), traditional regression, decision trees and k-nearest neighbours. The work presented in this thesis utilizes supervised learning where accelerometer data is the input and the health indicator is the output. Unsupervised learning on the other hand, does only require the input data. It learns to recognise features of relevance in the provided data. The autoencoder is an easy to understand concept, as it is in its purest form two mirrored neural networks representing an encoder part and a decoder part. The middle of the autoencoder is a layer of compressed features that can be used to replicate the data it has been trained on. When the autoencoder is trained, the input is the same as the output. The network will try to find the compressed features, which best link the input and output together with as little loss as possible. Loss is a concept that will be discussed later, but it is similar to general compression loss.

The concept of training a neural network can be seen as an optimization problem where the parameters to be optimized are the weights and the objective function is the loss. The loss is a measure of neural network training performance. A loss function is defined according to what the purpose of the network is. The most basic use of a loss function is basic error measurement. For instance, in a basic regression problem with a fixed amount of parameters, mean average error (MAE) or mean squared error (MSE) are good candidates for loss functions. In case of a multi-class classification problem, a more intricate loss function like categorical cross-entropy is required. Minimizing the loss is done through a process called backpropagation.

The core of backpropagation lies within a term called gradient descent. Gradient descent can be described by changing the trainable parameters and observing the change of loss. The amount of change per step is known as the learning rate. The word gradient comes from relating the change of each parameter with the respective change of loss, and descent comes from the minimization process of the loss. Figure 8 illustrates gradient descent. The height of the figure resembles the loss and the other two directions resemble the parameters. The blue curve represents the path of gradient descent.

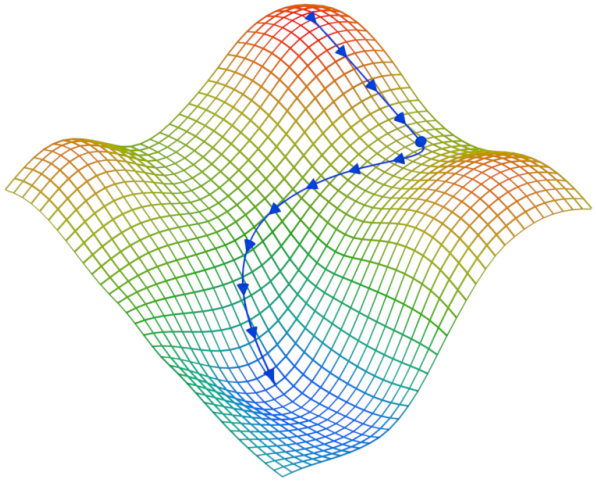

Figure 8: Visualized gradient descent

The backpropagation algorithm used in this work is called Adam. It is an efficient stochastic optimization method that computes individual adaptive learning rates for different parameters from estimates of the first and second moments of the gradients. The name Adam is derived from its description: **Ada**ptive **m**oment estimation. The algorithm is designed to combine the advantages of two other recent popular methods called AdaGrad (Duchi et al., 2011) and RMSProp (Tieleman & Hinton, 2012) [27].

The neural networks described in this thesis have a structure composed of several different types of layers. The layers existing between the input and output layers are often referred to as hidden layers. The most basic layer is a densely connected, or dense layer, consisting of a set of neurons as described earlier. There are also non-trainable layers in the structure. The function of these layers is to manipulate or reshape the outputs of other layers. The following layers make up the structure of our networks:

- Trainable layers
  - Densely connected layers
  - Convolutional layers
- Non-trainable layers
  - Pooling layers
  - Batch Normalization layers
  - Flatten layers
- Activation layers (for convenience)

3.2.1 Densely connected layers

The densely connected layer is the most fundamental layer in any neural network structure. It is made up of a fixed amount of neurons, each having weights from each neuron of the previous layer, to each neuron of the next layer. Considering Equation 13, any dense layer l in a neural network structure can be rewritten as a matrix product shown in Equation 14, assuming x^{l-1} and b are column vectors.

$$y^l = \sigma(W^l \cdot x^{l-1} + b^l) \tag{14}$$

The weight matrix of the current layer W^l is structured as follows where n^l and n^{l-1} show the amount of neurons in the current layer and the previous layer, respectively:

$$W^l = \begin{bmatrix} w_{11} & w_{12} & w_{13} & \dots & w_{1n^{l-1}} \\ w_{21} & w_{22} & w_{23} & \dots & w_{2n^{l-1}} \\ w_{31} & w_{32} & w_{33} & \dots & w_{3n^{l-1}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{n^l 1} & w_{n^l 2} & w_{n^l 3} & \dots & w_{n^l n^{l-1}} \end{bmatrix}$$

In practice, data flow through a network is represented by tensors. Tensors contain data of higher dimensions. The number of dimensions is known as the tensor rank. Rank 0, 1 and 2 tensors are scalars, vectors and matrices, respectively. The term tensors are generally used when the rank is 3 or greater. The neurons of a densely connected layer are always given by a vector, or rank 1 tensor.

3.2.2 Convolutional layers

A convolutional layer consists of a set of trainable kernels. These can be thought of as greater-than-zero rank tensor representations of a normal neurons. Most applications therefore include analyzing continuous 1D, 2D or even 3D data. The kernels act like filters on the input and produce filtered outputs. This is done through convolutional operations. Similar to normal dense layers, the kernels are trained in order to produce the filtered outputs best linking the network output to the input. The produced output filters are essentially equivariant to translation, thus if a feature is translationally shifted in the input, it will equally shifted in the output filter.

A convolutional operation across an infinite range is defined as follows [28]:

$$(f * g)(t) \triangleq \int_{-\infty}^{\infty} f(\tau) g(t - \tau) d\tau$$

where $f * g$ is a convolution between functions f and g. Rewriting this equation to work with finite discrete data gives Equation 15

$$(f * g)[n] = \sum_{m=-\frac{M}{2}}^{\frac{M}{2}} f[m] g[n - m] \tag{15}$$

where $(f * g)[n]$ is the resulting filtered value at kernel position n and M is the kernel size. Note that the output size will be M smaller than the input size due to the convolutional lack of edge padding.

Convolutional layers can have multiple channels. This is widely applied in image recognition networks where images have multiple color channels. Each channel is of the same size and the convolutional operations are done in parallel using the same kernels. The output filters are also multi-channel. The channel dimension is per definition just another dimension of the data tensors. However, the convolutional layers discretely interpret the dimension as separate channels. Figure 9 visualizes a one-dimensional, two-channel convolutional operation:

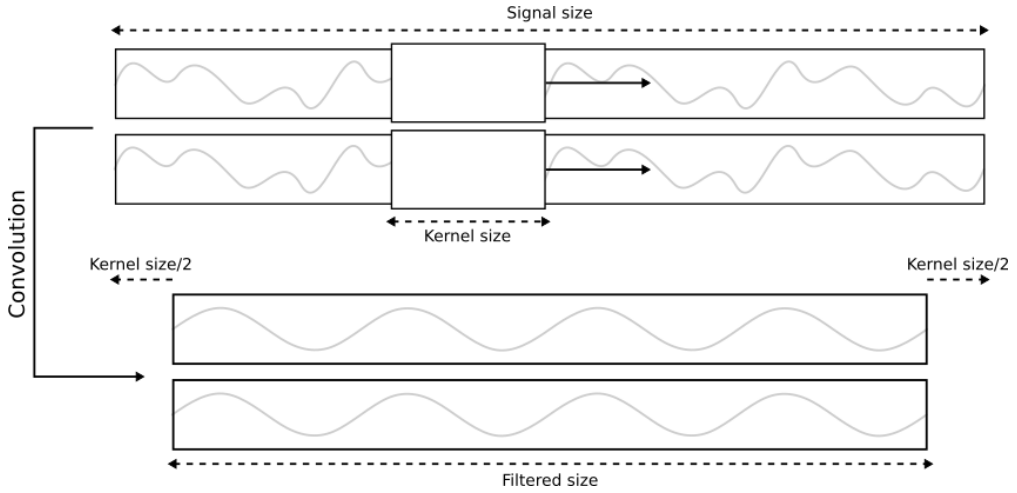

Figure 9: One-dimensional, two-channel convolution operation

Essentially, the idea behind convolutional layers is to recognize the features that are relevant to the regression or classification problem in question. These features can be visualized when a trained network is used in predictions, as will be shown later.

3.2.3 Pooling layers

Pooling is defined as a discretization process. However, since we are already working with discrete data, it can be defined as a sample-based discretization process. It can be compared to a windowed operation with strides equaling the window length, that returns a value per window based on a rule or set of rules. In convolutional neural networks, it is very common to implement the pooling technique called max-pooling. Max-pooling returns the largest values of each window, thus outputing a subsampled version of the input. Figure 10 shows an example of a two-dimensional max-pooling. The input is a $4 \times 4 \times 1$ (single channel) tensor, the pooling (window) size is 2×2, so the output becomes a $\frac{4}{2} \times \frac{4}{2} \times 1 = 2 \times 2 \times 1$ tensor.

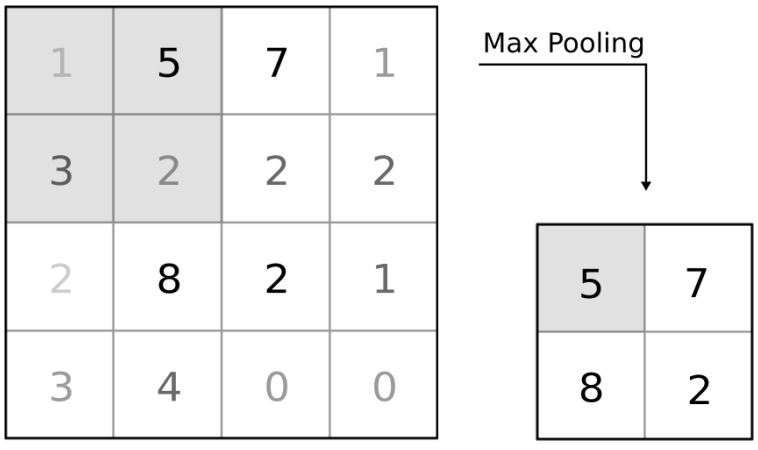

Figure 10: Max-pooling

The max-pooling is usually applied to the filtered output of the convolutional layers. There is a variety of reasons to do this. First and foremost, downsampling increases the performance of the network; less data

to process equals less prediction times. Secondly, combining convolutional layers with max-pooling layers results in a certain degree of translational invariance. As the convolutional layer outputs are equivariant to translation, the max-pooling layers will act as a receptive field picking up the relevant features even though they are translationally shifted. This is especially useful for processing signals subject to phase shifting and varying frequency, in this case variation in operation speeds. Ideally, it is desired to tune the size of the max-pooling windows to best suit the data. Smaller windows reduce the translational invariance, but larger windows may discard important features. Stacking multiple convolutional and max pooling layers can be a solution to this problem, but at the expense of performance reduction. It is consequently also harder to train deeper models [29].

Recently research has been made into developing convolutional neural networks without the application of max-pooling layers, but utilizing convolution strides instead [30]. This would reduce the network complexity, but not necessarily show better performance than using max-pooling, hence why it is not applied here.

3.2.4 Batch Normalization

Internal covariate shift is a problem that occurs as inputs to a layer are shifted during training. This results in slower training speeds by requiring smaller learning rates and more careful parameter initialization [29]. As earlier mentioned, normalizing the input to the network can help increasing convergence time during training. Batch Normalization is a technique that can be implemented at each layer of the network in order to speed up training by resolving the problem with internal covariate shift.

Batch normalization can be seen as a static layer, although in practice it needs some variable parameters to keep track of the distributions of each dimension. It is common to place this layer between trainable layers and their respective activations. For some (sigmoidal) activations, this will remove the phenomenon known as vanishing gradients, which will be explained later. Batch Normalization works by normalizing the outputs of the trainable layer in mini-batches. The mean becomes zero and the variance becomes one, hence why it needs to keep track of the distributions.

3.2.5 Flatten layer

The Flatten layer is essientially a layer that reshapes the input tensor into a "flat" tensor of rank 1, ie. vector. This is very useful for combining the output of a convolutional layer chain into a dense layer of neurons. Figure 11 shows an example of a two-dimensional, three-channel tensor being flattened into a one-dimensional vector:

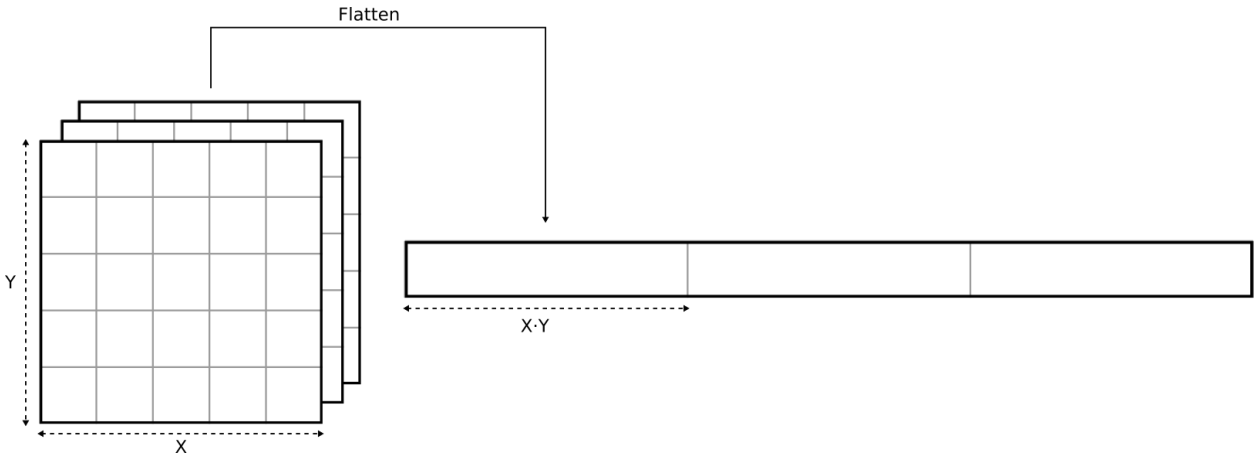

Figure 11: Rank three tensor flattening

3.2.6 Activations

As earlier mentioned, an activation is a function that outputs a value from a neuron based on the input to the neuron. However, activations can operate across layers as well. A good example of this is the softmax activation, that takes the values of each neuron in the layer as input.

The models presented here utilize two types of activations. The first is the rectified linear unit (ReLU) activation and the second one is the logistic sigmoid activation. The functions and their respective derivatives are plotted in Figure 12.

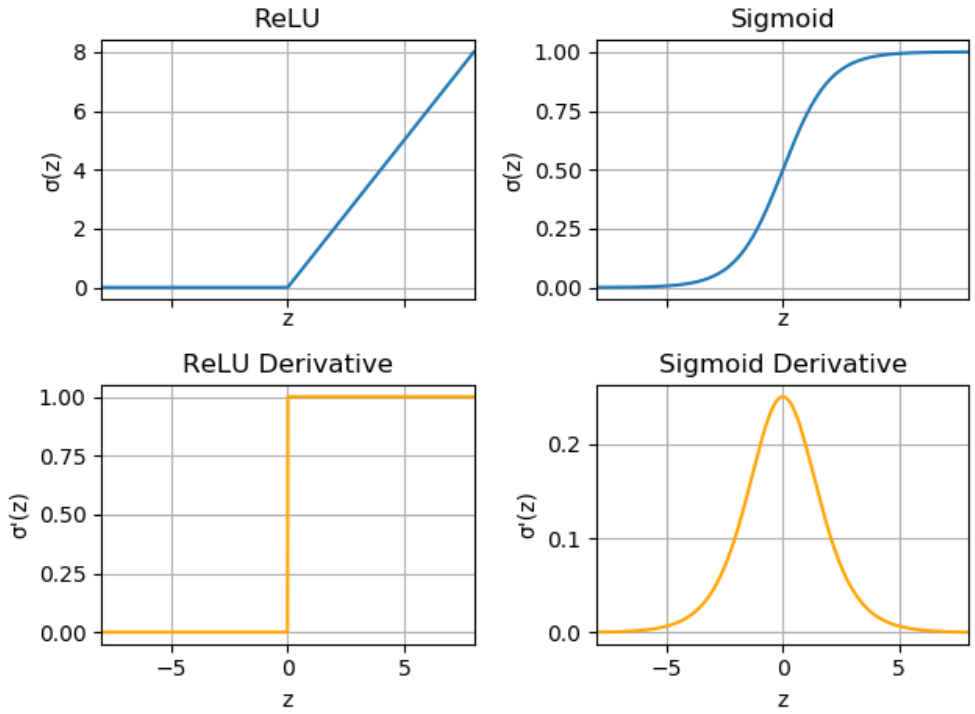

Figure 12: ReLU and Sigmoid activations

The ReLU activation is a linear unit function that only outputs positive values. It is defined in Equation 16:

$$\sigma(z) = \max(0, z) \tag{16}$$

This activation is used in the hidden layers of the models, with the advantage that it does not pose any threat of vanishing gradients like there are in traditional hyperbolic tangent or logistic sigmoid activation functions. In addition to this, it is proven to be an equal or even better biological representation than the latter, despite the hard non-linearity and non-differentiability at zero [31].

In the work presented here, the logistic sigmoid function is used as a special case in the output of the network models. It effectively forces the output values to lie withing the range of 0 to 1, which is ideal for the HI presented in Section 3.1.1. The sigmoid function is defined in Equation 17:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{17}$$

The bottom left graph of Figure 12 shows the derivative of a sigmoid function. Notice how quick the function converges towards $\sigma(z) = 0$ for $z < 0$ and $z > 0$. An activation gradient of 0 will make it impossible for

the gradient descent algorithm to approximate the parametric direction of change. This is the phenomenon known as vanishing gradients and is the reason sigmoid activation functions are not used in the hidden part of the networks.

3.2.7 CNN HI prediction models

This thesis will present two slightly different convolutional neural network models; one that takes time-series signals as input (Figure 13 and Table 1) and one that takes complex frequency spectrum data as input (Figure 14 and Table 2). These will be referred to as the first and the second networks, respectively.

The input to the first network is pre-worked time series data. One signal consists of 2560 sample points. Since convolutional layers work with data channels, the input tensor is second rank. The first convolutional layer has a kernel size of 16 and generates 64 output filters. The following max pooling layer has a window size of 16, reducing the amount of data by quite a lot. A batch normalization is applied in the next layer, followed by the ReLU activation. The second convolutional layer block is the same as the first one. The reason that the kernel and pooling sizes are big is that network should be able to generalize the variation in phase shift between signals, and the stretching of the signals due to different operating speeds (translational invariance). The flatten layer concatenates all the output filters from the second convolutional block. It is followed by the first and second densely connected layers, which are added to improve the overall learning capacity of the model. The outputs from both these densely connected layers are batch normalized and activated with a ReLU function. The last dense layer contains only one neuron, which is activated using the sigmoid function.

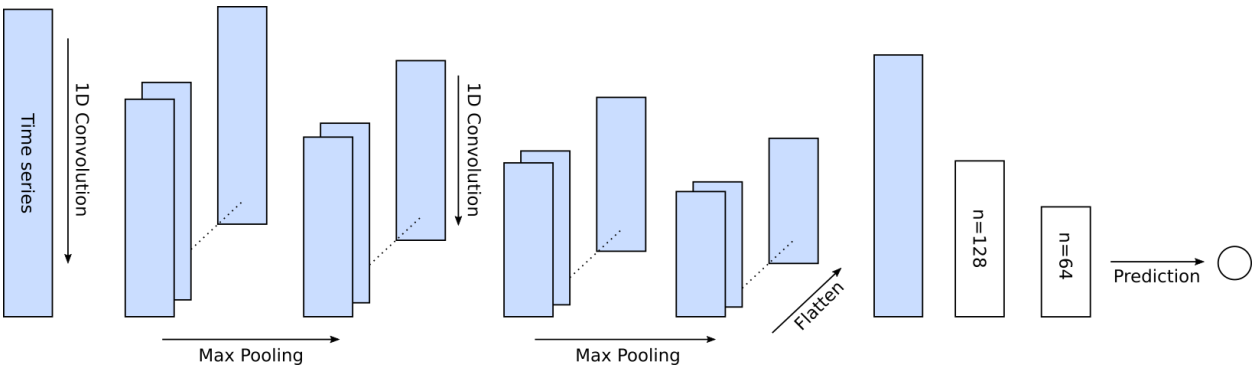

Figure 13: CNN structure with time series input

Layer	Output Shape	Number of parameters	Misc.
Input	2560, 1	-	
Convolutional 1D	2545, 64	1088	Kernel size 16
Max Pool 1D	159, 64	-	Pool size 16
Batch Normalization	159, 64	256	
ReLU Activation	159, 64	-	
Convolutional 1D	144, 128	131200	Kernel size 16
Max Pool 1D	9, 128	-	Pool size 16
Batch Normalization	9, 128	512	
ReLU Activation	9, 128	-	
Flatten	1152	-	
Dense	128	147584	128 neurons
Batch Normalization	128	512	
ReLU Activation	128	-	
Dense	64	8256	64 neurons
Batch Normalization	64	256	
ReLU Activation	64	-	
Dense	1	65	Output
Sigmoid Activation	1	-	

Table 1: CNN layers with time series input

The second network is very similar to the first network, except for an extra channel added to the input. Since only the positive frequencies are used, the first dimension will be of length 1281 ($\frac{2560}{2} = 1280$ plus a value at 0 Hz). The first channel of the second dimension contains the real FFT values, while the second channel contains the imaginary FFT values. The only other difference from the first network is the convolution kernel sizes and pooling sizes which are much lower. The kernel and pooling size of the first convolutional block are both 4. For the second convolutional block they are 8 and 4, respectively. As the signals are already transformed into the frequency domain, there is no need to compensate for the phase shift between time series signals. The only thing that matters is the stretching of the spectrum due to varying operation speeds, hence these values should be kept lower in this network.

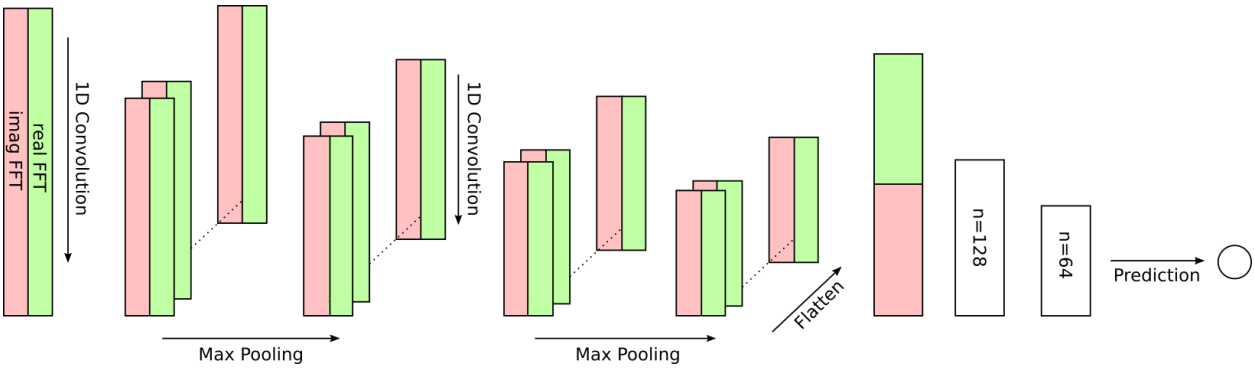

Figure 14: CNN structure with frequency spectrum input

Layer	**Output Shape**	**Number of parameters**	**Misc.**
Input	1281, 2	-	
Convolutional 1D	1278, 64	576	Kernel size 4
Max Pool 1D	319, 64	-	Pool size 4
Batch Normalization	319, 64	256	
ReLU Activation	319, 64	-	
Convolutional 1D	312, 64	32832	Kernel size 8
Max Pool 1D	78, 64	-	Pool size 4
Batch Normalization	78, 64	256	
ReLU Activation	78, 64	-	
Flatten	4992	-	
Dense	128	639104	128 neurons
Batch Normalization	128	512	
ReLU Activation	128	-	
Dense	64	8256	64 neurons
Batch Normalization	64	256	
ReLU Activation	64	-	
Dense	1	65	Output
Sigmoid Activation	1	-	

Table 2: CNN layers with spectrum input

4 Results and discussion

4.1 Datasets

The data used to train and test the models presented in this thesis comes from the IEEE PHM 2012 Data Challenge and is the result of experiments conducted on the PROGNOSTA platform [32]. A total of 17 unique datasets where recorded, containing vibration data from accelerometers mounted on a bearing, that was run-to-failure for each dataset. Each dataset is made up of files containing recordings of vibration signals. Vibration is recorded both vertically and horizontaly, although the work presented here only uses the horizontal data. The vibration signals are recorded every 10 seconds at a sampling frequency of 25.6 kHz. Each file contains 2560 data points, or 0.1 s of sampling. In addition to recording of vibration signals, bearing temperature was also recorded. However, the temperature data is not used in this work. The experiments ended when the vibration signal overpassed a g-force threshold of 20g.

The PROGNOSTA platform enables highly accelerated bearing degradation processes by enabling for a radial load up to 4000 N to be applied to the bearing. This also means that there is no need for seeded faults, so the experiments end with bearings containing most types of defects, of all bearing components. Figure 15 shows the difference between healthy and degraded bearing components.

Figure 15: Normal versus degraded bearing components [32]

PROGNOSTA's components consist of the asynchronous motor with gearbox, shafts, a shaft support bearing and a radial load applied by a pneumatic jack system. The inner race of the bearing is connected to the bearing support shaft. The pneumatic jack applies a radial load onto external ring of the shaft support bearing, amplified by a lever arm. The motor speed and direction, as well as the radial load can be controlled. An overview of the platform is shown in Figure 16.

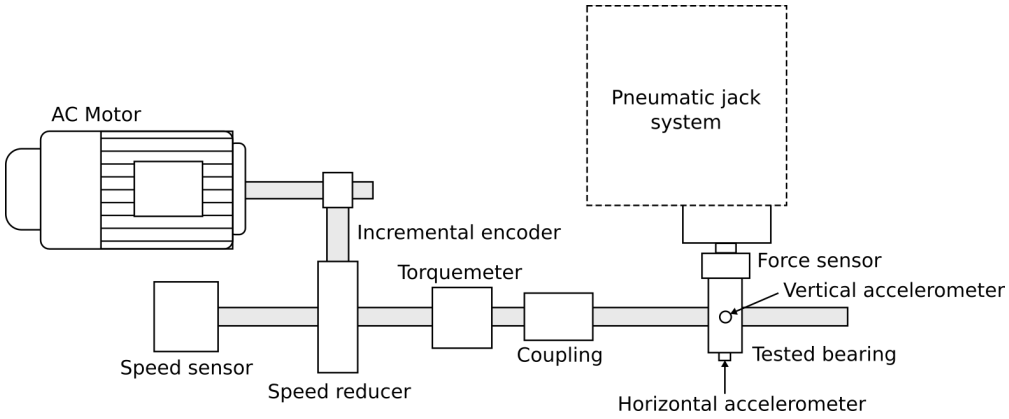

Figure 16: Overview of PROGNOSTIA

Table 3 shows the three different operating conditions of the experiments.

Condition	Speed	Load
1	1800 RPM	4000 N
2	1650 RPM	4200 N
3	1500 RPM	5000 N

Table 3: Operating conditions

Of the 17 different datasets, six were used to train the model, while the remaining eleven were used to validate the model after training. Table 4 shows how the datasets are distributed and their respective operating conditions. The naming convention of each dataset is "Bearing" followed by the operating condition number, then the experiment number of said condition.

	Condition 1	Condition 2	Condition 3
Training sets	Bearing1-1	Bearing2-1	Bearing3-1
	Bearing1-2	Bearing2-2	Bearing3-2
Validation sets	Bearing1-3	Bearing2-3	Bearing3-3
	Bearing1-4	Bearing2-4	
	Bearing1-5	Bearing2-5	
	Bearing1-6	Bearing2-6	
	Bearing1-7	Bearing2-7	

Table 4: Overview of the datasets and their operating conditions

Figure 17 shows the RMS-plots of each validation dataset. Each value is computed from the total RMS of a data file, using the definition of RMS shown in Equation 18. Some resemble a clean exponential growth, while others are contain a lot of noise.

$$f_{RMS} = \sqrt{\frac{1}{T_2 - T_1} \int_{T_1}^{T_2} [f(t)]^2 dt} \tag{18}$$

where f_{rms} is the rms value of function $f(t)$ across the interval $[T_1, T_2]$ [33].

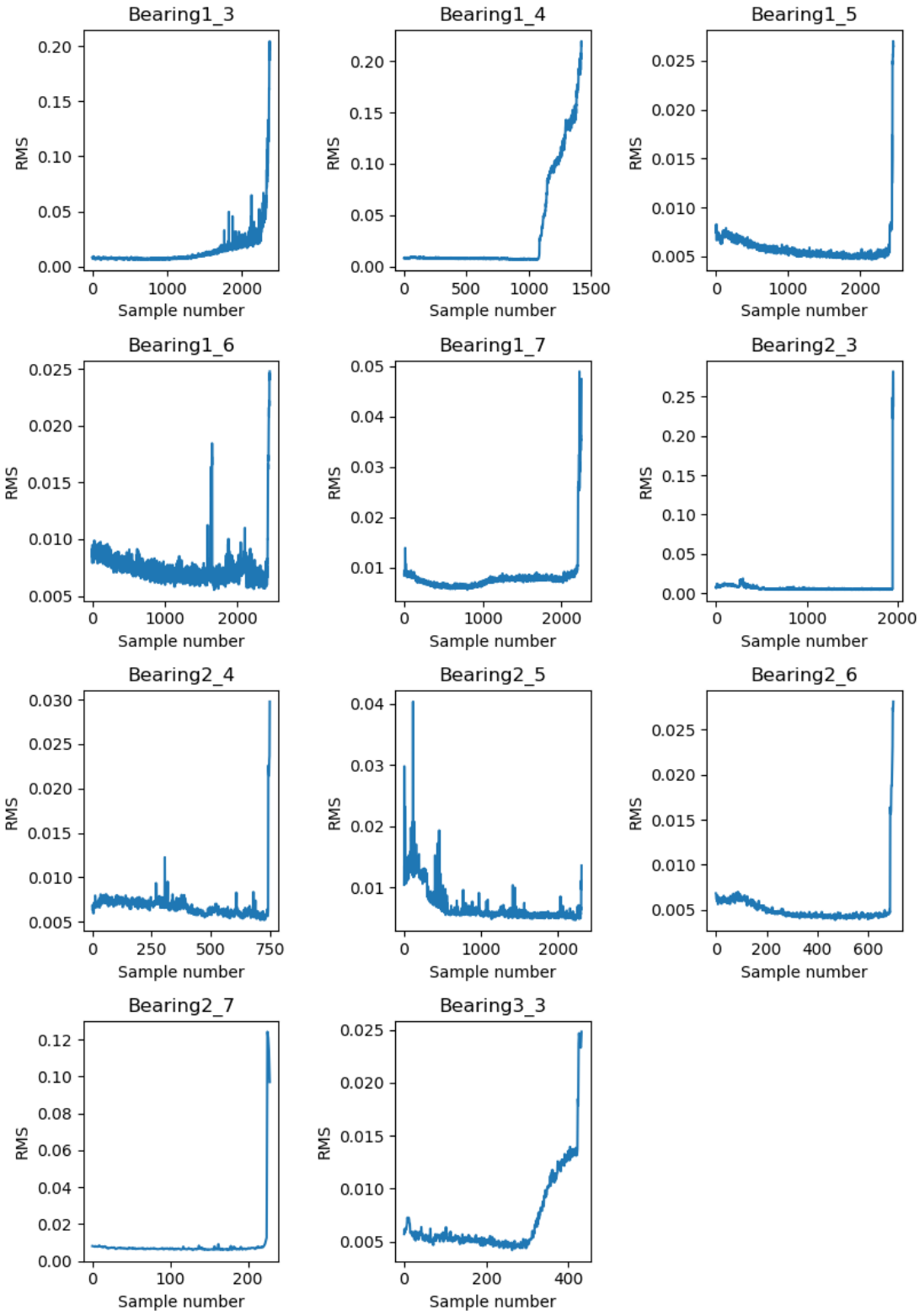

Figure 17: RMS-plots of validation datasets

4.2 Time series prediction model

The training history of the first network is shown in Figure 18. It is evident that the network has started converging already after the first epoch, with a training loss of below 20% MAE. The validation loss hits its most optimal point after just 5 epochs. The validation loss begins to increase after this point, while the training loss keeps decreasing. This is a major indicator of overfitting. A training callback initiates an early stopping after 10 epochs, and restores the model weights of the 5th epoch.

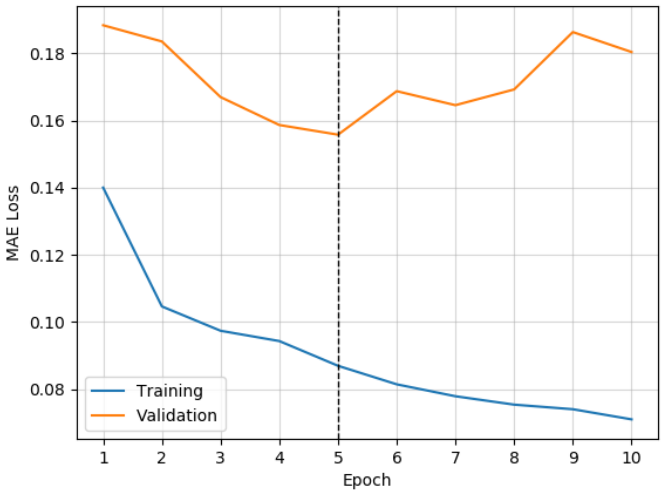

Figure 18: Time series model training history

Figure 19 shows the predicted HIs across all the validation datasets. Each prediction dot corresponds to the prediction of one time series vibration signal from the respective dataset. The linear blue line shows the optimal health indicator and the orange line shows a moving average with a window of 100 predictions.

The overall results are very mixed. Predictions of Bearing1-3 and Bearing2-6 show the overall best results, both having relatively low prediction spread and good linearity. A common feature of many of the datasets is that the predictions seem to increase quite linearly throughout the first half of the dataset, then start to diverge from the optimal HI. This could be due to the physical degradation being different to the training data, which in turn will result in different shapes of the vibration signals.

Bearing1-3 and Bearing2-6 also show that the translational invariance of the convolutional blocks work to a certain degree, as these two bearings were operated under different speeds. However, as Bearing3-3 shows poor predictions, one can question to what extent this translational invariance actually works, or if it is due to some other reason.

Some of the datasets, for instance Bearing2-3 and Bearing2-5, have a very large prediction spread. This is a sign that the network is struggling to predict those datasets accurately and may be a result of too much generalization of signals with $HI > 0.6$, as the spread usually is much smaller at lower values of HI. This might be the region where degradation usually start developing properly, thus signals from this point may greatly vary from one dataset to another.

The final thing worth noting is that in most cases, the network is able to recognize the early increase of HI in the datasets. Figure 17 reveals that the RMS-values of most datasets show very little change throughout most of the bearing lifetime. The network excels at recognising this early change, which is very good compared to the previous work presented in Section 1 where the condition classes were labeled based

on the RMS-plots.

Table 5 shows the scores of the predictions of the entires validation datasets. The linearity of the predictions w.r.t. the optimal HI is measured by the Pearson correlation coefficient, while the monotonicity is measured by the Spearman correlation coefficient. The MAE is the average value of the prediction MAEs throughout the dataset.

As previously concluded, Bearing1-3 and Bearing2-6 show the best scores, where Bearing1-3 have as much as 95.86% linear correlation as well as 95.90% monotonicity. The MAE is as low as 7.21%, making the overall prediction accuracy of this dataset 92.79%. Bearing2-7 show the highest MAE, whereas Bearing3-3 show both the lowest linear correlation and monotonicity with the scores 53.10% and 26.16%, respectively. The low (close to 0) monotonicity can be intuitively confirmed by noting that the curve of Bearing3-3 in Figure 19 is slightly declining throughout the larger portion the dataset.

Bearing Name	**Pearson**	**Spearman**	**MAE**
Bearing1-3	0.9586	0.9589	0.0721
Bearing1-4	0.7059	0.6585	0.1638
Bearing1-5	0.8435	0.8171	0.1236
Bearing1-6	0.7838	0.7836	0.2135
Bearing1-7	0.8490	0.8103	0.1151
Bearing2-3	0.7270	0.6659	0.1617
Bearing2-4	0.5829	0.8004	0.3102
Bearing2-5	0.6000	0.5110	0.1917
Bearing2-6	0.9331	0.9353	0.1014
Bearing2-7	0.7026	0.8268	0.3242
Bearing3-3	0.5308	0.2616	0.2616
Mean	**0.7470**	**0.7299**	**0.1659**

Table 5: Time series HI prediction scores

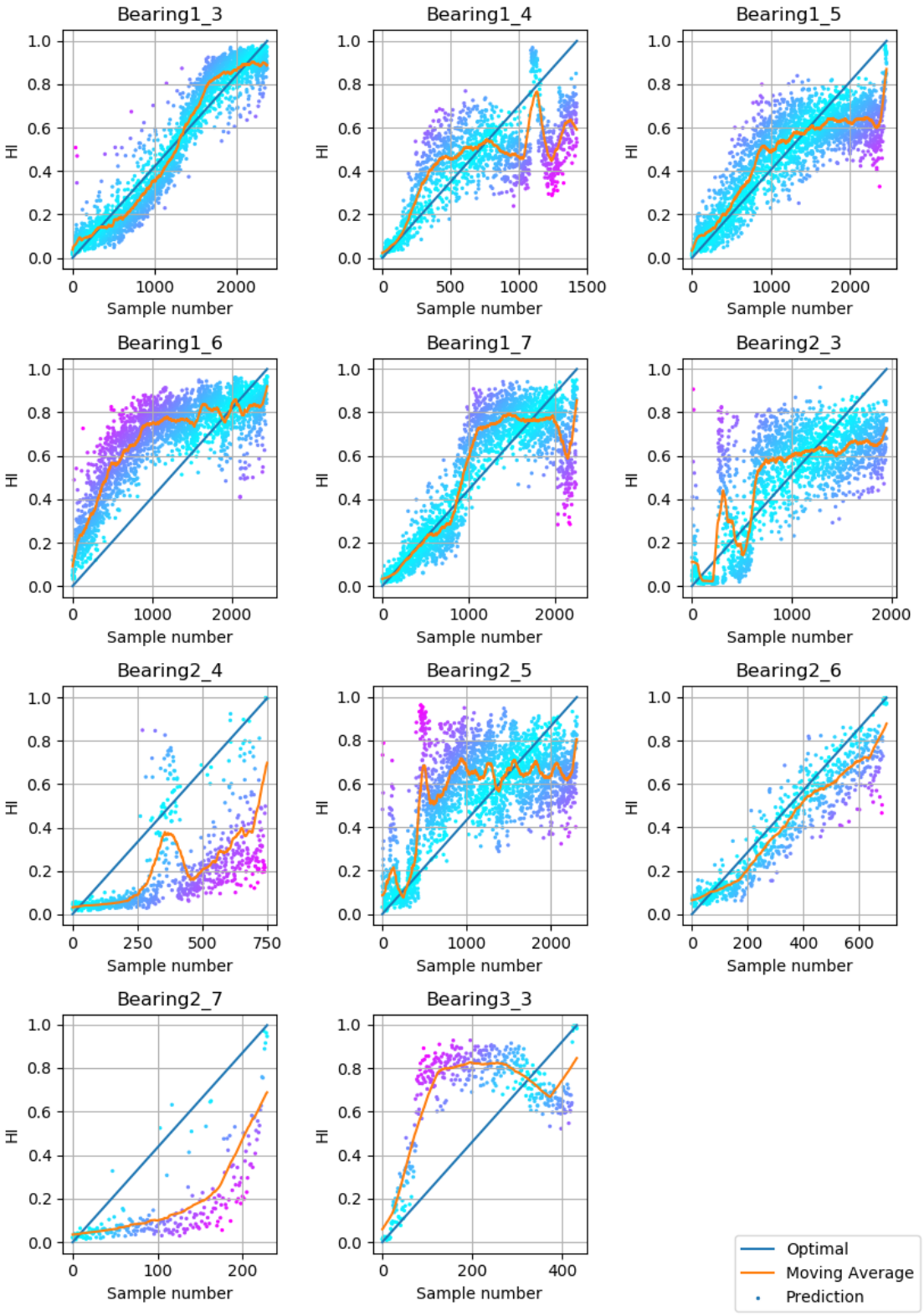

Figure 19: Time series HI predictions

Figures 20 and 21 show the the max-pooling outputs of the two convolutional blocks when predicted on a healthy signal (left) and a degraded signal (right) from Bearing1-3. The color values of each plot are normalized to better show the filter differences. The scaling is shown on the color bar to the right of each respective plot.

Beginning with the first convolutional block, note the difference in scaling between the healthy and degraded signals. This is expected due to the increased vibration RMS of a degraded bearing. Secondly it is worth noting that many of the filtered degraded signals suffer from little variation compared to the healthy one. This is most likely due to the vibration features being much more prominent in a healthy signal, compared to the noise pollution of a degraded signal. Thus, the convolutional kernels fit the healthy signals better than the noisy, degraded signals. However, some of the filtered degraded signals contain features that stand out. Filtered output 11 shows some positive peaks, while some of the outputs between 35 and 50 show some negative peaks. A lot of the filtered signals look similar, which means that the network could possibly be optimized by reducing the number of filters in this layer.

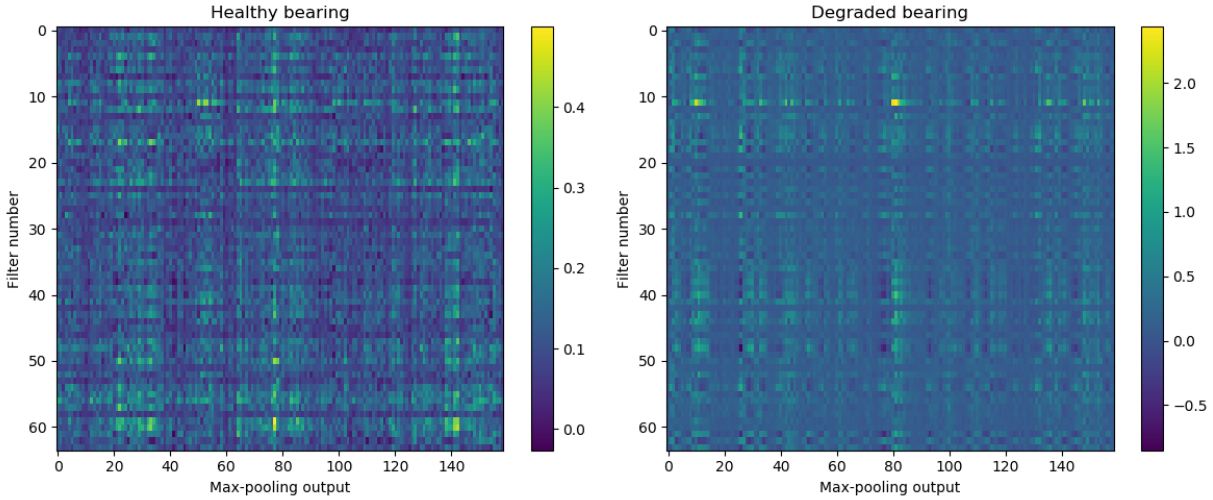

Figure 20: Pooling output of first convolutional block

The max-pooling outputs of the second convolutional block confirm the success of translational invariance. The values of each filtered output have little variance throughout the output, thus the phase shift and scaling of the vibration signals will not affect the output. In addition to this, the range of values are much higher for the degraded signal filters, therefore making it safe to assume that filtered values reflect the actual degradation.

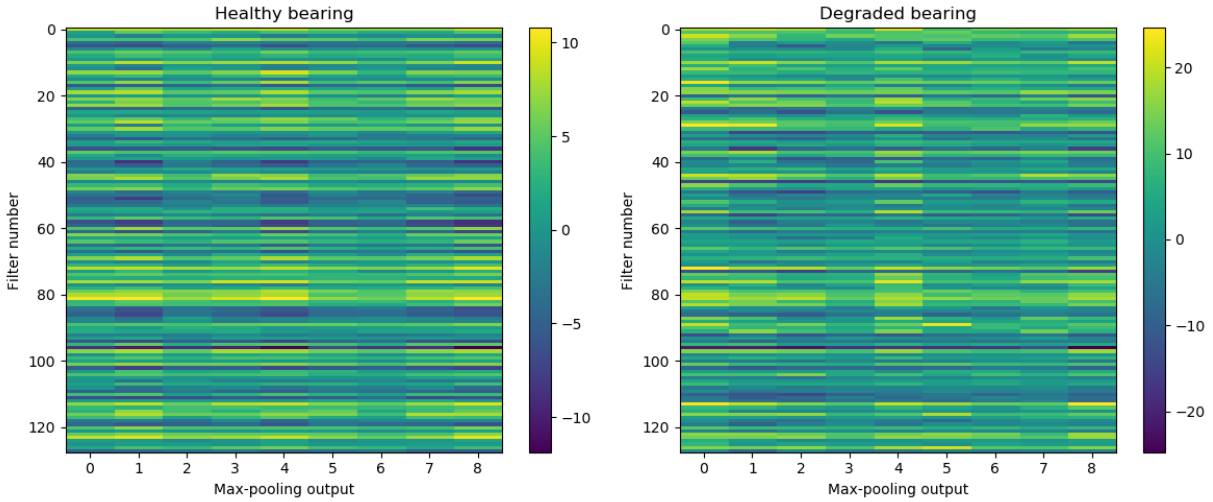

Figure 21: Pooling output of second convolutional block

4.3 Spectrum prediction model

The training history of the second network in Figure 22 shows that the model training validation has already finished converging during the first epoch. However, there is no clear overfitting during the next epochs even though the training loss keeps decreasing. The validation loss slightly dips below 14% MAE at epoch 13. The weights of this epoch are restored as the validation loss does not improve further during the next 5 epochs.

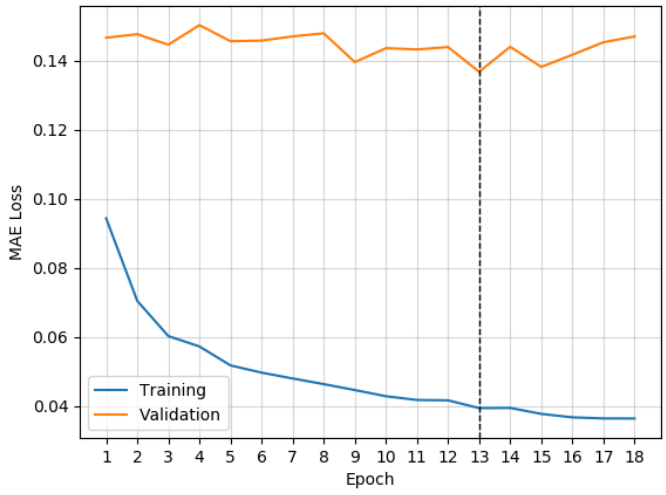

Figure 22: Spectrum model training history

The prediction results from the spectrum model in Figure 24 shows an overall slightly better performance. We can see the same characteristics of the time series model testify themselves in these results as well, however to a slightly lower degree in most cases. Bearing1-3 is performing approximately equally as good as the first model, while Bearing2-6 show worse results. On the other hand, Bearing1-4, Bearing1-5, Bearing1-6 show

some improvements. Bearing1-5, Bearing1-7, Bearing1-3 and Bearing3-3 show some improvement in the last portion of the datasets, where they are increasing more than in the first model. However, all predicted HI trends start slightly above 0, as was only the case with some of the results from the first model.

Common to the results of the first model is the bent shapes of some predicted HI trends. These are most likely reflections of the linear mapping of the exponential physical degradation done by the CNN model. Depending on type degradation, the shape of the actual physical degradation with respect to time will slightly change. When the model is trained to do the linear mapping for the HI, the difference in these physical degradation shapes is shown in the predicted trends from one dataset to another. This is illustrated in Figure 23.

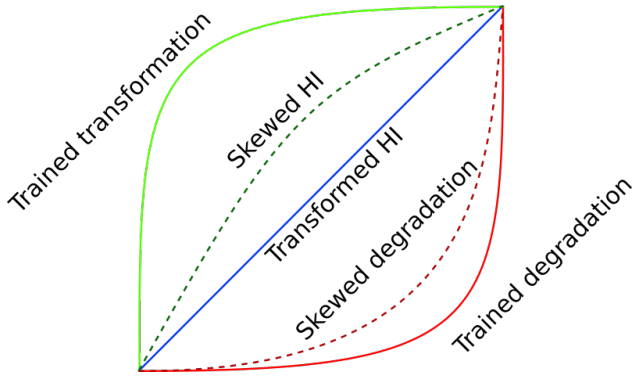

Figure 23: Mapping the HI

Table 6 show the scores of the predictions for the spectrum model. Predictions Bearing1-3 still show the best scores. However, Bearing2-6 show worse slightly worse scores. Otherwise, Bearing2-3 has improved while Bearing2-5 show increased linearity and monotonicity, but slightly higher MAE. This shows that MAE can be a bad scoring method when the prediction spread is high: Comparing the results of Bearing2-5 for the two models by looking at the predicted HI-trends shows that the second model is performing better overall. However, as the spread is much lower in the second model it punishes the MAE, since the HI trend is shifted away from the optimal HI throughout the first half of the dataset. In addition to this, if all the HIs of a dataset were predicted to be 50%, this would give much higher MAE than a shifted set of HIs with linear correlation of 100% w.r.t the optimal HI.

Bearing Name	**Pearson**	**Spearman**	**MAE**
Bearing1-3	0.9656	0.9680	0.0644
Bearing1-4	0.9037	0.9180	0.1047
Bearing1-5	0.8695	0.8582	0.1007
Bearing1-6	0.8921	0.8892	0.1158
Bearing1-7	0.9175	0.9016	0.0927
Bearing2-3	0.8277	0.8096	0.1227
Bearing2-4	0.8353	0.9318	0.2501
Bearing2-5	0.6507	0.5765	0.2113
Bearing2-6	0.8993	0.9257	0.1369
Bearing2-7	0.6877	0.9228	0.3254
Bearing3-3	0.8213	0.8703	0.2004
Mean	**0.8428**	**0.8702**	**0.1569**

Table 6: Spectrum HI prediction scores

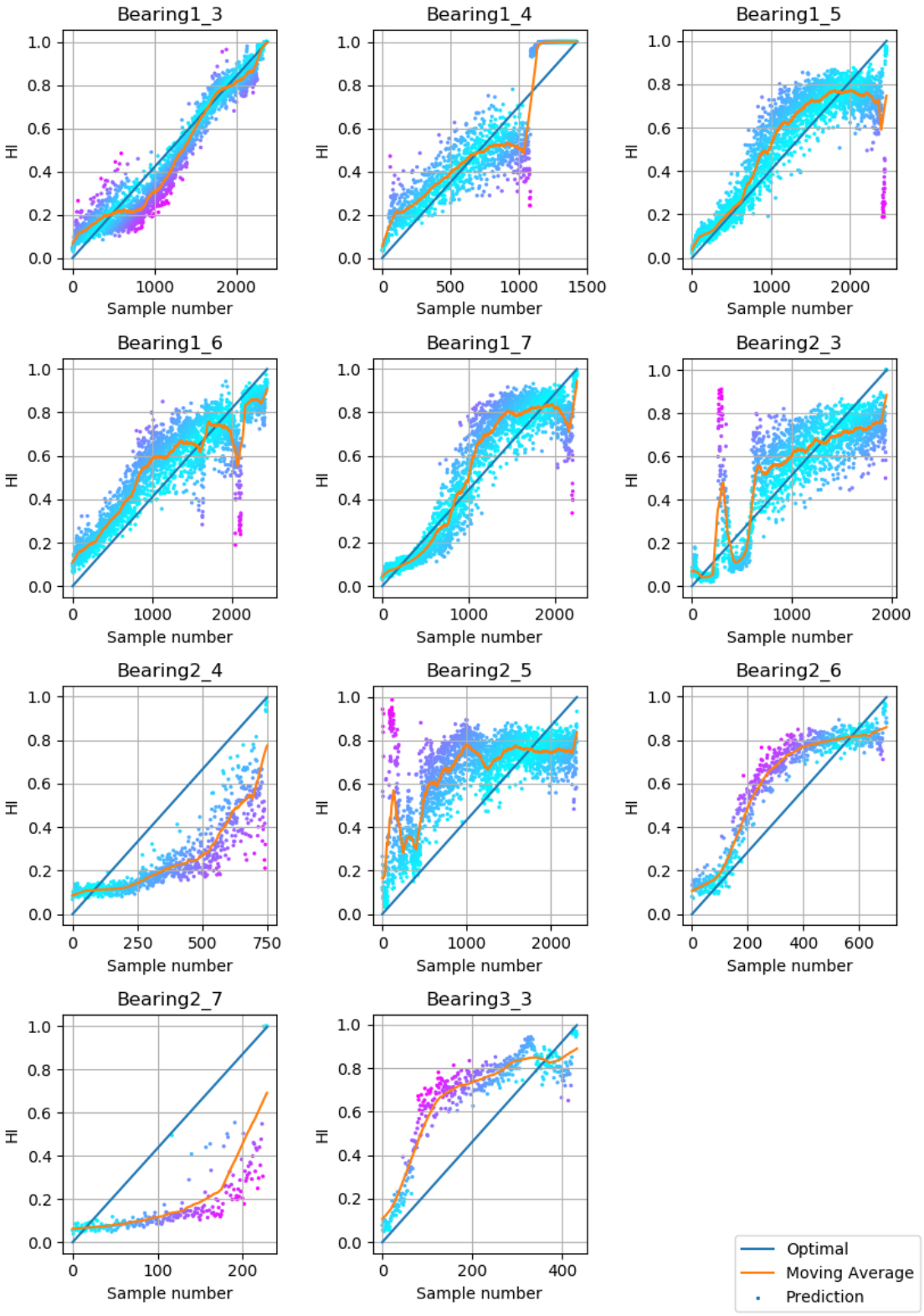

Figure 24: Spectrum HI predictions

4.4 RUL predictions

Figure 25 show the RUL predictions as functions of time for each dataset, based on the spectrum prediction model. The moving average RUL is calculated for each moving average HI using Equation 9 of Section 3.1.1. The linear estimation RUL is calculated from all the HIs up to the current point in time using the least squares method in the same section. The optimal RUL is the actual RUL throughout the dataset. Therefore, the best predicted RULs are the ones close to the optimal line at the RUL estimation time.

The moving average RULs all vary greatly in the beginning, as to be expected considering the denominator of Equation 9. Considering the predicted HIs of Figure 19, we can see that the RUL errors are heavily affected by errors in the predicted HIs. Also considering that many of the HI predictions were highly inaccurate towards the end of the datasets, the RUL errors are large at their most critical. The low early RUL certainty and large HI errors towards the end makes for a bad combination.

Table 7 show the actual RULs at the RUL estimation time versus the moving average RULs and the linear estimation RULs. The actual RULs are the ones defined in the PHM IEEE 2012 challenge.

We again see Bearing1-3 and Bearing2-6 performing very well based on moving average, however they perform worse based on the linear estimation, with Bearing2-6 predicted to have failed already 233 seconds ago. Bearing2-3 also performed good based on moving average. However, considering the HI trends it can be discussed to what extent some of these predictions were not just lucky. As for the linear estimation RULs, only Bearing1-5 to Bearing2-3 performed somewhat okay relative to the actual RUL. These predictions were still low compared to the best moving average RULs. However, the linear estimation RULs show a more smooth and natural decrease towards the end of the datasets.

Bearing Name	**Actual RUL [s]**	**Moving Average RUL [s]**	**Linear Estimation RUL [s]**
Bearing1-3	5730	5172.41	8204.70
Bearing1-4	339	238.56	9302.93
Bearing1-5	1610	9114.08	645.62
Bearing1-6	1460	3831.14	924.07
Bearing1-7	7570	3880.27	4059.74
Bearing2-3	7530	6794.24	4509.11
Bearing2-4	1390	8063.61	10359.88
Bearing2-5	3090	7110.97	-1146.21
Bearing2-6	1290	1304.77	-232.978
Bearing2-7	580	6906.17	6699.0
Bearing3-3	820	692.47	-417.58

Table 7: Predicted RULs

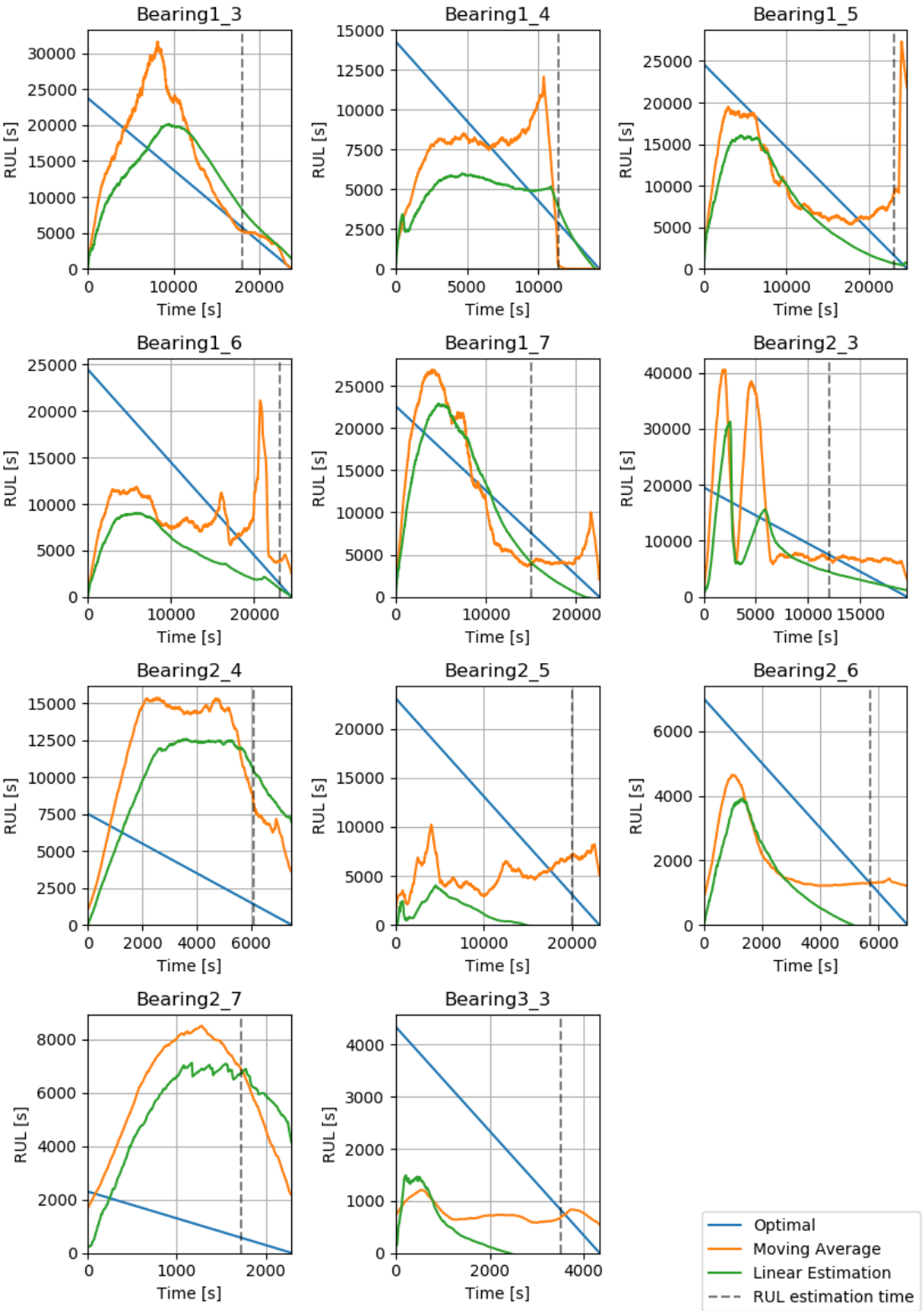

Figure 25: RUL predictions

The IEEE PHM 2012 Data Challenge provided a scoring ruleset for evaluating RUL predictions at a certain point in each of the validation datasets. A percentage error of RUL prediction of dataset i is calculated by

$$\%Er_i = 100 \times \frac{RUL_{i,true} - RUL_{i,pred}}{RUL_{i,true}} \tag{19}$$

where $RUL_{i,true}$ is the true RUL and $RUL_{i,pred}$ is the predicted RUL of the dataset. Early predictions are logically weighted more than late predictions, meaning that predictions with $\%Er_i > 0$ gives a better score than predictions with $\%Er_i < 0$. The scoring function A_i is defined as follows

$$A_i = \begin{cases} \exp^{-\ln(0.5)\cdot(Er_i/5)} & \text{if } Er_i \leq 0 \\ \exp^{+\ln(0.5)\cdot(Er_i/20)} & \text{if } Er_i > 0 \end{cases} \tag{20}$$

The plot of this function is shown in Figure 26. The final total score is calculated by taking the mean of each dataset, ie.

$$\text{Final Score} = \frac{1}{11}\sum_{i-1}^{11}(A_i) \tag{21}$$

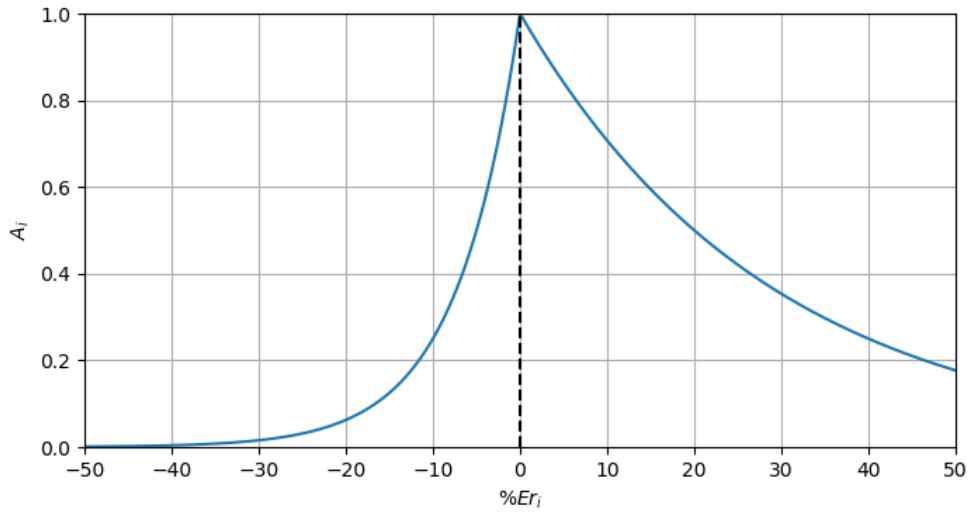

Figure 26: RUL prediction error scoring function

Table 8 shows the final scores. Although the mean moving average score is relatively high, many of the estimated datasets have a score of 0.

Bearing Name	**Moving Average Score**	**Linear Estimation Score**
Bearing1-3	0.7176	0.0024
Bearing1-4	0.0416	0.0078
Bearing1-5	0.0	0.1265
Bearing1-6	0.0	0.2802
Bearing1-7	0.1851	0.2010
Bearing2-3	0.7157	0.2497
Bearing2-4	0.0	0.0
Bearing2-5	0.0	0.0086
Bearing2-6	0.7647	0.0166
Bearing2-7	0.0	0.0
Bearing3-3	0.5833	0.0054
Mean	**0.2734**	**0.0817**

Table 8: Final RUL score results

5 Conclusion

Although the results are overall varied, the best predictions show that a model can be trained to estimate HIs for unique datasets. However, due to the worse performing predictions, it is not a reliable way to estimate HIs accurately. The final RUL estimations are overall poor, although a few resulted in high scores.

Judging by the good predictions in at least one dataset of all the different operating conditions, it is safe to assume that the models have a high degree of both load and speed independence. The filtered outputs of the convolutional blocks also confirm this for the time-series model. However, there are difficulties in training the network to do a linear transformation of an exponential degradation that should work for all degradation processes.

On the other hand, the models are very good at detecting early growth. Comparing to the RMS-plots of the datasets show that the models are able to distinguish between vibrational signals where the RMS values seem random, do not change, or even decrease. They do however seem to lose accuracy towards the last parts of the run-to-failure experiments.

The models can otherwise be used for diagnosis, ie. if the predicted HI exceeds a set threshold. A threshold set to $HI = 0.7$ would be triggered in most of the cases seen here, and approximately 30% RUL would be a good remaining lifetime to give an incipient fault warning.

Future work into this specific methodology would focus on improving and optimizing the CNN structures, as there is still potential in this area. It would also be interesting to look into robust regression methods for filtering out prediction outliers. However, conducting controlled experiments with seeded faults could open for a new methodology where the convolutional neural network models are trained as specific diagnosis and prognosis hybrids. A model trained on data with known faults could in theory learn to estimate the type of faults, if multiple, as well as the respective severity of each fault - all from vibration data.

References

[1] Subrata Karmakar, Surajit Chattopadhyay, Madhuchhanda Mitra, and Samarjit Sengupta. "Induction Motor and Faults". In: Springer, Singapore, 2016, pp. 7–28. DOI: 10.1007/978-981-10-0624-1{_}2. URL: http://link.springer.com/10.1007/978-981-10-0624-1_2.

[2] Fabio Immovilli, Claudio Bianchini, Marco Cocconcelli, Alberto Bellini, and Riccardo Rubini. "Bearing Fault Model for Induction Motor With Externally Induced Vibration". In: *IEEE Transactions on Industrial Electronics* 60.8 (Aug. 2013), pp. 3408–3418. ISSN: 0278-0046. DOI: 10.1109/TIE.2012.2213566. URL: http://ieeexplore.ieee.org/document/6269996/.

[3] Lucia Frosini. "Monitoring and Diagnostics of Electrical Machines and Drives: a State of the Art". In: *IEEE WEMDCD*. 2019.

[4] Linxia Liao and Felix Köttig. "A hybrid framework combining data-driven and model-based methods for system remaining useful life prediction". In: *Applied Soft Computing* 44 (July 2016), pp. 191–199. ISSN: 1568-4946. DOI: 10.1016/J.ASOC.2016.03.013. URL: https://www.sciencedirect.com/science/article/pii/S1568494616301223.

[5] Pawel Rzeszucinski, Maciej Orman, Cajetan T. Pinto, Agnieszka Tkaczyk, and Maciej Sulowicz. "Bearing Health Diagnosed with a Mobile Phone: Acoustic Signal Measurements Can be Used to Test for Structural Faults in Motors". In: *IEEE Industry Applications Magazine* 24.4 (July 2018), pp. 17–23. ISSN: 1077-2618. DOI: 10.1109/MIAS.2017.2740463. URL: https://ieeexplore.ieee.org/document/8344408/.

[6] Lucia Frosini, Marco Magnaghi, Andrea Albini, and Giovanni Magrotti. "A new diagnostic instrument to detect generalized roughness in rolling bearings for induction motors". In: *2015 IEEE 10th International Symposium on Diagnostics for Electrical Machines, Power Electronics and Drives (SDEMPED)*. IEEE, Sept. 2015, pp. 239–245. ISBN: 978-1-4799-7743-7. DOI: 10.1109/DEMPED.2015.7303696. URL: http://ieeexplore.ieee.org/document/7303696/.

[7] E. Martinez-Montes, L. Jimenez-Chillaron, J. Gilabert-Marzal, J. Antonino-Daviu, and A. Quijano-Lopez. "Evaluation of the Detectability of Bearing Faults at Different Load Levels Through the Analysis of Stator Currents". In: *2018 XIII International Conference on Electrical Machines (ICEM)*. IEEE, Sept. 2018, pp. 1855–1860. ISBN: 978-1-5386-2477-7. DOI: 10.1109/ICELMACH.2018.8507224. URL: https://ieeexplore.ieee.org/document/8507224/.

[8] Rodney K. Singleton, Elias G. Strangas, and Selin Aviyente. "The Use of Bearing Currents and Vibrations in Lifetime Estimation of Bearings". In: *IEEE Transactions on Industrial Informatics* 13.3 (June 2017), pp. 1301–1309. ISSN: 1551-3203. DOI: 10.1109/TII.2016.2643693. URL: http://ieeexplore.ieee.org/document/7795228/.

[9] D. Raul Quintero, W. Mejia, and J. A. Rosero. "Good practice for Electric Discharge Machining (EDM) bearing currents measurement in the induction motor and drives system". In: *2013 International Electric Machines & Drives Conference*. IEEE, May 2013, pp. 1384–1390. ISBN: 978-1-4673-4974-1. DOI: 10.1109/IEMDC.2013.6556317. URL: http://ieeexplore.ieee.org/document/6556317/.

[10] Bach Phi Duong, Sheraz Ali Khan, Dongkoo Shon, Kichang Im, Jeongho Park, Dong-Sun Lim, Byungtae Jang, and Jong-Myon Kim. "A Reliable Health Indicator for Fault Prognosis of Bearings." In: *Sensors (Basel, Switzerland)* 18.11 (Nov. 2018). ISSN: 1424-8220. DOI: 10.3390/s18113740. URL: http://www.ncbi.nlm.nih.gov/pubmed/30400203%20http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC6263687.

[11] I-Hsi Kao, Wei-Jen Wang, Yi-Horng Lai, and Jau-Woei Perng. "Analysis of Permanent Magnet Synchronous Motor Fault Diagnosis Based on Learning". In: *IEEE Transactions on Instrumentation and Measurement* 68.2 (Feb. 2019), pp. 310–324. ISSN: 0018-9456. DOI: 10.1109/TIM.2018.2847800. URL: https://ieeexplore.ieee.org/document/8400570/.

[12] Chuang Sun, Meng Ma, Zhibin Zhao, and Xuefeng Chen. "Sparse Deep Stacking Network for Fault Diagnosis of Motor". In: *IEEE Transactions on Industrial Informatics* 14.7 (July 2018), pp. 3261–3270. ISSN: 1551-3203. DOI: 10.1109/TII.2018.2819674. URL: https://ieeexplore.ieee.org/document/8325506/.

[13] Wenjun Sun, Rui Zhao, Ruqiang Yan, Siyu Shao, and Xuefeng Chen. "Convolutional Discriminative Feature Learning for Induction Motor Fault Diagnosis". In: *IEEE Transactions on Industrial Informatics* 13.3 (June 2017), pp. 1350–1359. ISSN: 1551-3203. DOI: 10.1109/TII.2017.2672988. URL: http://ieeexplore.ieee.org/document/7862893/.

[14] Arild Bergesen Husebø, Huynh Van Khang, and Witold Pawlus. "Diagnosis of Incipient Bearing Faults using Convolutional Neural Networks". In: *IEEE WEMDCD*. 2019.

[15] B K N Rao, P Srinivasa Pai, and T N Nagabhushana. "Failure Diagnosis and Prognosis of Rolling - Element Bearings using Artificial Neural Networks: A Critical Overview". In: *Journal of Physics: Conference Series* 364.1 (May 2012), p. 012023. ISSN: 1742-6596. DOI: 10.1088/1742-6596/364/1/012023. URL: http://stacks.iop.org/1742-6596/364/i=1/a=012023?key=crossref.be76dedb95a705fe11c46ee6a32036b8.

[16] P. Paris and F. Erdogan. "A Critical Analysis of Crack Propagation Laws". In: *Journal of Basic Engineering* 85.4 (Dec. 1963), p. 528. ISSN: 00219223. DOI: 10.1115/1.3656900. URL: http://FluidsEngineering.asmedigitalcollection.asme.org/article.aspx?articleid=1431537.

[17] *Paris Law Theory*. URL: https://www.fose1.plymouth.ac.uk/fatiguefracture/tutorials/FractureMechanics/Fatigue/FatTheory1.htm.

[18] Naipeng Li, Yaguo Lei, Jing Lin, and Steven X. Ding. "An Improved Exponential Model for Predicting Remaining Useful Life of Rolling Element Bearings". In: *IEEE Transactions on Industrial Electronics* 62.12 (Dec. 2015), pp. 7762–7773. ISSN: 0278-0046. DOI: 10.1109/TIE.2015.2455055. URL: http://ieeexplore.ieee.org/document/7154471/.

[19] Jerome L Myers, A (Arnold) Well, and ProQuest (Firm). *Research design and statistical analysis*. 2nd ed. Vol. 48. 08. Mahwah, N.J. : Lawrence Erlbaum Associates, 2013, pp. 48–4538. ISBN: 0805840370 (acid-free paper). DOI: 10.5860/choice.48-4538. URL: https://ebookcentral.proquest.com/lib/qut/detail.action?docID=474601.

[20] *numpy.linalg.lstsq — NumPy v1.16 Manual*. URL: https://docs.scipy.org/doc/numpy/reference/generated/numpy.linalg.lstsq.html.

[21] Arvind Neelakantan, Luke Vilnis, Quoc V. Le, Ilya Sutskever, Lukasz Kaiser, Karol Kurach, and James Martens. "Adding Gradient Noise Improves Learning for Very Deep Networks". In: (Nov. 2015). URL: http://arxiv.org/abs/1511.06807.

[22] Yann A. LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. "Efficient BackProp". In: Springer, Berlin, Heidelberg, 2012, pp. 9–48. DOI: 10.1007/978-3-642-35289-8{_}3. URL: http://link.springer.com/10.1007/978-3-642-35289-8_3.

[23] James W. Cooley and John W. Tukey. "An algorithm for the machine calculation of complex Fourier series". In: *Mathematics of Computation* 19.90 (May 1965), pp. 297–297. ISSN: 0025-5718. DOI: 10.1090/S0025-5718-1965-0178586-1. URL: http://www.ams.org/jourcgi/jour-getitem?pii=S0025-5718-1965-0178586-1.

[24] Eric W. Weisstein. "Fast Fourier Transform". In: (). URL: http://mathworld.wolfram.com/FastFourierTransform.html.

[25] Yoshua Bengio. "Practical recommendations for gradient-based training of deep architectures". In: (June 2012). URL: http://arxiv.org/abs/1206.5533.

[26] Warren S. McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The Bulletin of Mathematical Biophysics* 5.4 (Dec. 1943), pp. 115–133. ISSN: 0007-4985. DOI: 10.1007/BF02478259. URL: http://link.springer.com/10.1007/BF02478259.

[27] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: (Dec. 2014). URL: http://arxiv.org/abs/1412.6980.

[28] Eric W. Weisstein. “Convolution”. In: (). URL: http://mathworld.wolfram.com/Convolution.html.

[29] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: (Feb. 2015). URL: http://arxiv.org/abs/1502.03167.

[30] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. “Striving for Simplicity: The All Convolutional Net”. In: (Dec. 2014). URL: http://arxiv.org/abs/1412.6806.

[31] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. “Deep Sparse Rectifier Neural Networks”. In: *JMLR W&CP* 15 (June 2011), pp. 315–323. ISSN: 1938-7228. URL: http://proceedings.mlr.press/v15/glorot11a.html.

[32] Patrick Nectoux, Rafael Gouriveau, Kamal Medjaher, Emmanuel Ramasso, Brigitte Chebel-Morello, Noureddine Zerhouni, and Christophe Varnier. *PRONOSTIA: An experimental platform for bearings accelerated degradation tests*. June 2012, pp. 1–8.

[33] Eric W. Weisstein. “Root-Mean-Square”. In: (). URL: http://mathworld.wolfram.com/Root-Mean-Square.html.

A Tools and practicalities

Python 3.6.5 is used in conjunction with a list of external libraries to produce the results shown in Section 4. Table 9 show the libraries used with their respective usage areas:

Library	**Usage**
Numpy	General computing and data structures
Scipy	Statistical functionality
Pandas	Reading datasets
Keras	Neural network functionality and model definition
Tensorflow	Backend for Keras
Matplotlib	Plotting figures and graphs

Table 9: Python libraries

The main source code material developed for the project can be found at `https://github.com/arhusebo/rul-based-hi`.

A.1 Data feed

The data feed is done the same way as described in Section 3.1.5. In practice, a TimeDataSequence class that inherits from the Keras Sequence class is used. This class is similar to a normal Python generator object, except that Keras is able to recognize relevant information, eg. total amount of batches, during training and prediction of models. The TimeDataSequence parameters of relevance are

- *metadata*
- *purpose*
- *health_index*
- *batch_size*
- *noise_scale*
- *normalize*
- *shuffle*

where metadata is a Python dictionary containing information about the datasets, purpose is either train or test, health_index is a function of HI taking a dataset description dictionary, current file index and dataset length as parameters, batch_size is the number of time series signals per batch, noise_scale is a factor of signal standard deviation to be added as gaussian noise to the signal, normalize is a boolean whether to normalize the signal or not and finally shuffle which is a boolean value whether to shuffle the datasets or not. The latter should be on for training generators, and off for prediction generators.

When the TimeDataSequence object is instantiated, IDs are generated based on whether to use the training or testing data described in the metadata dictionary. If shuffling is enabled, these IDs are shuffled. This shuffling also happens at the end of each training epoch. When a model, during training or prediction, asks for the next batch, the TimeDataSequence object finds the IDs at the current position and loads the signals corresponding to the respective IDs into memory. They are then labeled based on the health indicator function provided during object instantiation. Finally the signals have noise applied and are normalized if enabled.

For providing spectrum data, a SpectrumDataSequence class is made that inherits the TimeDataSequence class. When a SpectrumDataSequence object is asked to generate a batch, its parent first creates a batch of time series data. Each sample of this batch has its complex spectrum calculated using the FFT. The labels stay the same. The spectrum data batch is then returned.

The metadata dictionary is loaded from a json file and is structured as followed:

- path to datasets superdirectory
- file length
- default column separator
- train data
  - path to relative dataset subpath
  - channel
  - specific column separator
- test data
  - path to relative dataset subpath
  - channel
  - specific column separator

A.2 Training

Depending on the model, a function returning a Keras representation of said model is imported. A training Sequence object (purpose set to train, shuffle on) and a validation Sequence object (purpose set to test, shuffle off) is instantiated with their desired parameters. The Keras network model provides a function for fitting it using a Sequence object. The function takes the amount of training epochs, validation Sequence and miscellaneous callbacks as parameters. The early stopping callback is provided here. After the model is successfully trained, the structure and weights are saved to disc.

A.3 Validation

The previously trained model is loaded. A loop is initiated, looping over each dataset provided in the metadata test datasets. The following happens for each dataset: A new Sequence object is instantiated with the purpose set to test and shuffle turned off. The batch size is also set to 1. The loaded model provides a predict on Sequence object function, which is used to predict on the data provided by the validation data Sequence. The true values are extracted by looping through the data Sequence again. Finally, statistics are calculated based on the results, and figures are plotted.