A Comprehensive Methodology for Intelligent On-board Condition Monitoring of Electric Drives

Dissertation

der Mathematisch-Naturwissenschaftlichen Fakultät

der Eberhard Karls Universität Tübingen

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

(Dr. rer. nat.)

vorgelegt von

MSc. Ivan Melendez Vazquez

aus Puebla, Mexiko

Tübingen

2024

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualifikation:	16.09.2024
Dekan:	Prof. Dr. Thilo Stehle
1. Berichterstatter	Prof. Dr. Oliver Bringmann
2. Berichterstatter	Prof. Dr. Lars Hedrich

Abstract

Electric drives are often used to power electrified vehicles such as electric cars or pedelecs. As any other mechanical system, electric drives wear out with time, which, consequently, increases the possibility of sudden failures that might compromise the safety of people in the surroundings of the vehicle powered by the damaged drive. A method to effectively increase the reliability of electric drives is the on-board condition monitoring of the device. By applying condition monitoring, failures can be detected and classified in advance, which allows to take preventive measures before a failure occurs.

This thesis introduces a comprehensive methodology which comprises all relevant stages for the development of intelligent algorithms for condition monitoring: from the data acquisition and pre-processing to the systematic generation of accurate and compact machine learning models for timely preventive measures. The main objective of the introduced approach is to predict the remaining useful life of an electric drive. Furthermore, the presented approach also enables the on-board fault diagnosis if required. The proposed methodology is validated with a proprietary database consisting of data collected with electric drives for pedelecs. The findings of this case study show that the introduced methodology enables a reliable identification of the end of useful life of electric drives, which is required for data labeling. Moreover, the generated models are able to accurately predict the end of useful life of an electric drive with sufficient time in advance, and to correctly identify the damaged element as well. Finally, this study shows how the resulting models are suitable for an embedded implementation for on-board condition monitoring due to their reduced size and computational complexity. In a second case study, the failure prognosis algorithms are compared with other related techniques using a benchmark database. This study demonstrates the high efficiency of the proposed method to estimate the time to failure with very compact models, when compared to other related approaches.

Kurzfassung

Elektroantriebe werden häufig für den Antrieb elektrisch betriebener Fahrzeuge wie Elektroautos oder Pedelecs verwendet. Wie jedes andere mechanische System nutzen sich elektrische Antriebe mit der Zeit ab. Dies erhöht folglich die Wahrscheinlichkeit plötzlicher Ausfälle, die die Sicherheit von Personen in der Umgebung des von dem beschädigten Antrieb angetriebenen Fahrzeugs gefährden könnten. Eine Methode, um die Zuverlässigkeit von Elektroantrieben effektiv zu erhöhen, ist die Zustandsüberwachung des Geräts an Bord. Auf diese Weise können Ausfälle im Voraus erkannt und klassifiziert werden, was es ermöglicht, vorbeugende Maßnahmen zu ergreifen, bevor es zu einem Ausfall kommt.

Diese Arbeit stellt eine umfassende Methodik vor, die alle relevanten Phasen für die Entwicklung intelligenter Algorithmen für die Zustandsüberwachung umfasst: von der Datenerfassung und Vorverarbeitung bis zur systematischen Generierung genauer und kompakter maschineller Lernmodelle für rechtzeitige Präventivmaßnahmen. Das Hauptziel des vorgestellten Ansatzes ist die Vorhersage der Restnutzungsdauer eines Elektroantriebs. Darüber hinaus ermöglicht die Methodik bei Bedarf auch die On-Board-Fehlerdiagnose. Eine proprietäre Datenbank, bestehend aus Daten, die aus einem elektrischen Antrieb für Pedelecs erfasst wurden, ermöglichte die Validierung der vorgeschlagenen Methodik. Unsere Ergebnisse zeigen, dass die Methodik eine zuverlässige Bestimmung des Endes der Nutzungsdauer von Elektroantrieben ermöglicht, was für die Datenkennzeichnung erforderlich ist. Darüber hinaus sind die generierten Modelle in der Lage, das Ende der Nutzungsdauer eines elektrischen Antriebs mit viel Zeit im Voraus genau vorherzusagen und auch das beschädigte Element korrekt zu identifizieren. Schließlich eignen sich die resultierenden Modelle aufgrund ihrer reduzierten Größe und Berechnungskomplexität für eine eingebettete Implementierung zur Zustandsüberwachung an Bord. In einer zweiten Fallstudie wurde dieser Ansatz mit verbunden Techniken unter Verwendung einer Benchmark-Daten-bank verglichen. Diese Studie zeigt die hohe Effizienz der vorgeschlagenen Methode zur Abschätzung der Zeit bis zum Versagen mit sehr kompakten Modellen im Vergleich zu anderen Ansätzen.

Acknowledgments

I would like to express my deepest gratitude to my supervisor, Prof. Dr. Oliver Bringmann, for his exceptional guidance, insightful feedback, and unwavering support throughout the development of this doctoral thesis.

I am also deeply thankful to my mentor at Bosch, Dr. Rolando Dölling, whose trust in my potential has been a constant source of confidence and motivation. His invaluable feedback and practical insights have been instrumental in refining my work.

A heartfelt thanks to my mother, whose love, encouragement, and tireless efforts laid the foundation for my academic journey. I owe much of my success to her.

I would also like to honor the memory of my father, whose influence and values continue to inspire me. Though he is no longer with us, his wisdom and integrity have left an indelible mark on my life, guiding me even in his absence.

Most of all, I am profoundly grateful to my wife, Cecilia, whose steadfast support has been my greatest source of strength throughout this journey. Her patience, understanding, and unwavering belief in me during the highs and lows of this process have been indispensable. Her love and encouragement were the driving forces that helped me persevere to the very end.

Contents

Chapter 1

Introduction

The electrification of means of transport like cars and bicycles has found in recent years a tremendous growth. Electrified vehicles have two main components, the battery, which is the energy source, and the electric drive, which powers the vehicle. As any other mechanical system, electric drives wear out with the use. This raises the probability of a malfunction with time. A sudden failure of a drive's mechanical element represents a high risk for the driver and the people in the surroundings of the vehicle or system, which is powered by the damaged electric drive. Usually, to reduce this risk, a *preventive maintenance* policy is implemented. This strategy defines specific service intervals at which the maintenance of the equipment needs to take place, even if the device does not show any anomaly that affects its operation. However, this approach generates additional costs by performing unnecessary repairs in components that could still operate safely. Moreover, these constant intervals might be too long in certain circumstances. In that case, the system could break down before the next service takes place, jeopardizing the safety of the users. One method to improve the reliability of the maintenance time selection is achieved through the constant monitoring of the system's condition. In this way, the service takes place when an abnormal condition has been monitored. Thereby, unnecessary repairs and unexpected failures are minimized.

Initially, condition monitoring of mechanical systems was performed by placing determined sensors, mainly accelerometers, in strategical positions inside of machinery. The collected raw signals were transformed into frequency spectra and thresholds were determined for relevant frequencies. A malfunction was detected once a threshold was exceeded. Furthermore, a diagnosis of the component responsible for the fault could be done by performing an order analysis. This method for monitoring the system's condition required a deep knowledge about the system dynamics. Moreover, many false alarms were caused by sudden external disturbances, which influenced the sensors measurements. Towards the beginning of the 2000s, traditional signal processing techniques for condition monitoring were complemented with intelligent algorithms, which were able to automatically build the correlation between a given input and output, without the need to build a complex mathematical model. Moreover, these algorithms improved the efficiency of sole signal processing techniques by facilitating the fusion of multiple sensors to find new patterns from the extracted features. A second generation of approaches focused on the implementation of a predictive maintenance policy, which core element is the estimation of the time to failure of a system. In this strategy, the information generated by the sensors that monitor the system is used to identify undesired trends at an early stage. Thereby, the optimal service time can be defined in advance and the resources required to perform the maintenance can be properly prepared. The estimation of the remaining useful life is a more complex task than the fault isolation. It can hardly be solved with traditional machine learning models, especially if multiple faults may arise or the system operates under changing conditions. On the other hand, the availability of computers with high performance enabled the use of deep learning algorithms to accurately diagnose faults and forecast the time of failure of a system. One disadvantage of deep learning architectures is the need of large databases to be efficiently trained. The acquisition of big databases for the development of failure prognosis models is a highly complex task. This requires to collect data from several devices from their healthy state until they break down. Typically, mechanical systems can operate hundreds of hours before any sign of degradation can be perceived by the sensors. Thus, the collection of a reliable database for health prognosis can easily take several months and even years. To cope with these challenges, benchmark databases have been created with data generated during simulations or experimental setups. The data of these benchmark databases has served to develop and validate multiple approaches for failure prognosis. Since the state-of-the-art techniques for failure prognosis are developed only with experimental data and without a specific application in mind, they focus only on increasing the accuracy of the models. However, the increased efficiency comes with the cost of very high computational complexity. This hinders the model deployment in embedded systems for certain applications such as the on-board monitoring of mobile systems powered by an electric drive. Moreover, no method considered an efficient strategy to collect the data required for the training of deep learning models. The data collection has to be joined by a method to efficiently label the gathered data as well.

The aim of this work is to develop an efficient comprehensive methodology, which covers the strategy of data collection, processing and labeling, together with the training of monitoring algorithms and their optimization for embedded implementation. Moreover, a strategy to face the challenges that arise with the application of the algorithms in real driving circumstances, such as varying conditions, has to be enclosed within this comprehensive methodology. The focus of this approach is to enable the on-board condition monitoring of electric drives, especially drive units of pedelecs. The research question and the requirements of the methodology are introduced in the following section.

1.1 Research question and requirements

The following research questions are derived in pursuit of the development of a comprehensive methodology that enables a reliable on-board condition monitoring of electric drives: How can the learning models be optimized to suit into an embedded system without a considerable accuracy degradation? How can the key elements of the condition monitoring tasks be shared to simultaneously forecast and identify a failure with the lowest computation complexity? How can the flexibility of the approach be guaranteed, independently of the configuration of the electric drive? Which is an optimal strategy to obtain enough data that enables the generation of intelligent algorithms for failure prognosis? How can the data be easily labeled for the failure forecasting task? Based on these questions, the criteria that the comprehensive methodology has to fulfill are clustered in three groups: the model performance, the embedded implementation and the flexibilitiy of the methodology. Furthermore, a fourth category of requirements that are related to the application is given.

Requirement 1 - Model performance

Requirement 1.1 - Prognosis and diagnosis: Although the methodology focuses on the estimation of the remaining useful life of electric drives, it should provide the guidelines that enable the creation and deployment of learning models for both failure prognosis and diagnosis.

Requirement 1.2 - High reliability for prognosis: It is essential to minimize late predictions of failures. Moreover, false alarms should also be kept to a low level.

Requirement 2 - Suitable for an embedded implementation

Requirement 2.1 - Compact models: To enable the deployment of the trained learning algorithms into an embedded system, it is essential to build compact models with a reduced amount of parameters that fit into the system's memory.

Requirement 2.2 - Low computational complexity: The computation of the condition monitoring algorithms can be performed with a time separation which can vary from a few minutes to several hours. Therefore, a very low performance is not crucial. Nonetheless, the higher the amount of multiply-accumulate operations, the greater the memory space required to store the partial computations. To reduce the memory load, a model with low computational complexity is required, i.e. models with less than 1 million multiply-accumulate operations.

The target embedded system throughout this work is a ST Nucleo-F411RE with a Cortex-M4 processor, which is suitable for models with low memory and low computational complexity [59].

Requirement 3 - Flexibility

Requirement 3.1 - High flexibility: The flexibility refers to the capability of an approach to be implemented for the condition monitoring of different electric drives with a low effort.

Requirement 3.2 - Suitable for unlabeled data: A technique to easily and effectively label the data for prognosis tasks has to be considered within the developed approach.

Requirement 4 - Application oriented

Requirement 4.1 - Use of internal sensors only: The methodology is thought for electric drives that are used in mass produced systems such as electric cars or pedelecs.

Therefore, only the information provided by the sensors that already exist inside the drive should be used. By considering this, additional costs due to the extra sensors and their corresponding measuring devices can be avoided.

1.2 Thesis outline

In this section, a short overview of each chapter that constitutes this work is given. Moreover, it is indicated which part of the research or results are covered by the published papers.

First, chapter two provides the theoretical background required for the development and understanding of the methodology for on-board condition monitoring. Furthermore, a short description of electric drives is given at the beginning of this chapter.

The third chapter introduces related approaches for fault diagnosis and failure prognosis. Both machine learning and deep learning methods are introduced. The research of the state-of-the-art regarding condition monitoring lets us derivate the research questions, which are analyzed within this work. The last part of this chapter presents a summary of the requirements that are covered by existing approaches.

The comprehensive methodology for on-board condition monitoring is presented in chapter four. First, a strategy to collect the data required for the development of the intelligent algorithms is given. Then, the manual feature engineering method and signal processing techniques are described. Afterwards, two novel approaches to reduce the dimensionality for classification and regression tasks are detailed. Next, the approach for data labeling, which was presented in the second publication, is detailed. Moreover, slight modifications of this labeling method are presented in this thesis to improve the efficiency of this approach. Afterwards, two intelligent algorithms for failure prognosis are introduced: a Gaussian Process for Regression, and a novel deep learning architecture, presented in the third publication, which is named Multipath Temporal Convolutional Network. Additionally, the method to automatically select the optimized hyperparameters of the MTCN with a genetic algorithm is described. This method was presented in the third publication as well. Then, the fault diagnosis approach performed with a multiclass Support Vector Machine is introduced. This diagnosis approach follows the research of the first publication. At the end of this chapter, optimization strategies that can be implemented to deploy the trained models in an embedded system are given.

In chapter five, two case studies with the data collected from pedelec drive units is presented. These studies enabled the validation of each component of the proposed comprehensive methodology for the on-board monitoring of electric drives. The results of the first and second case studies were published in the second and first publications respectively. Chapter six introduces a numerical comparison between related approaches for failure prognosis and the algorithms introduced in this work, using a benchmark database. These results were published in the third publication.

Finally, chapter seven summarizes this thesis and provides an outlook about the future research necessary to improve the proposed algorithms for on-board condition monitoring.

1.3 Publications

Within the development of this research, the following works have been published in conference proceedings:

- I. M. Vazquez, R. Doelling and O. Bringmann, ”Fault Diagnosis Approach for Pedelec Drive Units Based on Support Vector Machines,” 2019 International Conference on Control, Automation and Diagnosis (ICCAD), Grenoble, France, 2019, pp. 1-6.
- I. Melendez, R. Doelling and O. Bringmann, ”Self-supervised Multi-stage Estimation of Remaining Useful Life for Electric Drive Units,” 2019 IEEE International Conference on Big Data (Big Data), Los Angeles, CA, USA, 2019, pp. 4402-4411.
- I. Melendez-Vazquez, R. Doelling and O. Bringmann, ”Multipath Temporal Convolutional Network for Remaining Useful Life Estimation,” 2020 IEEE International Conference on Big Data (Big Data), Atlanta, GA, USA, 2020, pp. 4137-4146.

Moreover, the following patent has been published at the German patent office:

- I. Melendez-Vazquez and R. Doelling, “Verfahren zum Ermitteln eines Zustands eines elektrischen Antriebs eines Fahrrads, Computerprogramm, Speichermedium und Fahrrad,” German Patent DE102019203816

Chapter 2

Foundations

First, this chapter provides the description of fault-diagnosis terminology. Then, a description of the electric drives, their main elements and type of failures is given. This chapter also introduces signal processing techniques and machine learning algorithms. The mathematical foundations given in this chapter are useful to describe and understand the proposed condition monitoring methodology for electric drives, which is introduced in Chapter 4.

2.1 Fault diagnosis terminology

These terms are defined within the *Reliability, Availability and Maintainability* dictionary and in German norms such as DIN standards. A summary of the definitions is introduced in [35]. A **fault** is an unpermitted deviation of at least one characteristic property of the system from the aceptable/ standard condition. A **failure** is a permanent interruption of a system's ability to perform a required function under specified operating conditions. Moreover, **fault detection** is the task to determine whether there is a fault present in a mechanism; **fault isolation** focuses on the identification of the fault root cause; **fault identification** quantifies the magnitude of the fault; and **fault diagnosis** comprises the detection, isolation and identification of a fault.

2.2 Electric drives

An electric drive is a mechatronic system, which task is the conversion of electrical energy into mechanical energy [41]. Electric drives consist of an electric motor, a transfer mechanism, a power electronic converter, a measurement system, and a control system [49]. In Figure 2.1, a generalized structure of an electric drive is depicted.

Almost every electric motor configuration, either linear or rotational, can be used in an electric drive. The mechanical transmission consists of shafts, gears, clutches and bearings. Its task is to increase the mechanical power at the output shaft. Furthermore, the power electronic converter refers to a network of semiconductor power switches. They supply the motor with power, which is regulated by the controller. The measurement

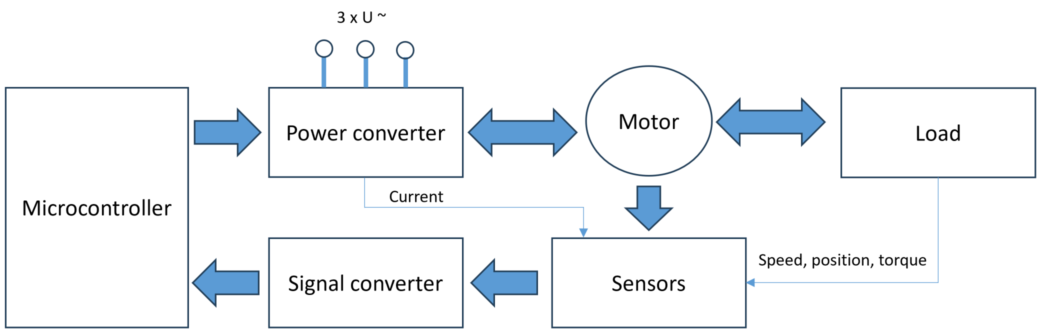


Figure 2.1: Generalized structure of an electric drive [49].

system is composed of sensors that measure the currents, voltages, torque, temperature, etc. In the control system or drive controller, the signal processing, the analogue-digital conversion and the motor control take place. The controller can be a microcontroller, an FPGA or a computer. Finally, the electrical power can be supplied from the alternative current grid, or from a direct current source such as a battery.

The diagnosis and forecasting is performed mostly for the mechanical elements of the drive, i.e. the mechanical transmission and the electric motor components. As seen in Figure 2.2, the mechanical elements have three strages in their life cycle, which have different failure probabilities. Within the first operations cycles, many elements tend to fail due to manufacturing or assembly errors. Afterwards, the systems operate for long periods under normal conditions, in which random failures ocasionally arise due to excesive load, corrosion or other factors [103]. In the last stage of their life, the mechanical elements have a progressive condition worsening, which increases the failure probability.

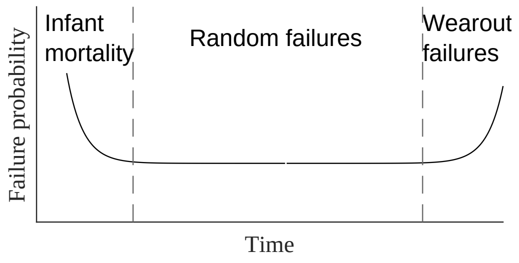


Figure 2.2: Probability of a failure over time (Bathtub curve).

This thesis focusses only on the mechanical wear failures, which can be forecasted. On the other hand, the electronic elements such as resistors, capacitors, inductors, etc., suffer sudden failures that cannot be monitored with much time of anticipation. Moreover, due to the complexity of most electronic systems, it is almost impossible to trace the faulty spot once the failure has occurred [102].

2.3 Signal processing techniques

2.3.1 Descriptive statistics

Descriptive statistics are a helpful method to reduce the available data to a manageable amount that can be easier analyzed and explained to a variety of stakeholders [20].

Central measures

The sample mean $\bar{x}$ is one of the most useful mathematical descriptions of a dataset. This metric can be interpreted as the central value of a discrete time series x and it is given by (2.1), where N is the vector size of the time series x.

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i. \tag{2.1}$$

Another useful central metric is the root-mean-square (RMS) value, given by:

$$RMS = \sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2}. \tag{2.2}$$

Moments about the mean

The statistical moments about the mean are used to describe the shape of distribution curves [60]. The second moment is known as the sample variance $var(x)$, which is a metric of the dispersion around the mean, and is given by:

$$var(x) = \frac{1}{N}\sum_{i=1}^{N} (x_i - \bar{x})^2, \tag{2.3}$$

this metric changes the magnitude of the measurement due to the squared difference. The standard deviation (σ) restores the amplitude of the variance to the original data distribution and is defined for a sample of a population as:

$$\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N} (x_i - \bar{x})^2}. \tag{2.4}$$

The skewness (η), the third moment about the mean, contains the information about the symmetry of a distribution [100]. If there is a greater concetration of data points to the right of the mean, the skewness is negative and the distribution is *skewed to the left*. On the other hand, if the mode of the data set is less than its mean value, then the skewness is positive and the data distribution is *skewed to the right*. If the distribution is symmetrical, the skewness is zero. This coefficient is given as follows:

$$\eta = \frac{1}{N}\frac{\sum_{i=1}^{N}(x_i - \overline{x})^3}{\sigma^3}. \tag{2.5}$$

The kurtosis (κ) is the fourth moment about the mean. It measures how closely the data are distributed around the mode and is given by:

$$\kappa = \frac{1}{N}\frac{\sum_{i=1}^{N}(x_i - \overline{x})^4}{\sigma^4}. \tag{2.6}$$

Measures of correlation

Given two time series x_i and x_j, the sample covariance matrix is given by:

$$q_{jk} = \frac{1}{N}\sum_{i=1}^{N}(x_{ij} - \overline{x_j})(x_{ik} - \overline{x_k}), \tag{2.7}$$

which is a measure of the correlation between both time series. The sample covariance is:

$$Q = \frac{1}{N}\sum_{i=1}^{N}(x_i - \overline{x})(x_i - \overline{x}). \tag{2.8}$$

Another correlation metric is the *Pearson's* coefficient ρ_{XY}, which is given by:

$$\rho_{XY} = \frac{q_{xy}}{\sigma_x \sigma_y}. \tag{2.9}$$

Furthermore, the Mahalanobis distance indicates the similarity of a vector y regarding a base distribution X and is given by [11]:

$$d = \sqrt{(y - \overline{X})^T Q^{-1}(y - \overline{X})}, \tag{2.10}$$

where $\overline{X}$ is the distribution's mean and Q is the sample covariance. Contrary to the Euclidean distance (2.27), which only measures the distance between two points, the Mahalanobis distance considers the correlation among the variables in the computation through the inverse convariance matrix. If the features are highly correlated, the values of the covariance matrix are high and the resulting Mahalanobis distance is decreased. In Fig. 2.3, an example of two sample points that have a similar Euclidean distance r to the distribution center but different Mahalanobis distance MD is displayed. Through this example, it can be visualized the effectiveness of this metric as an outlier detection method.

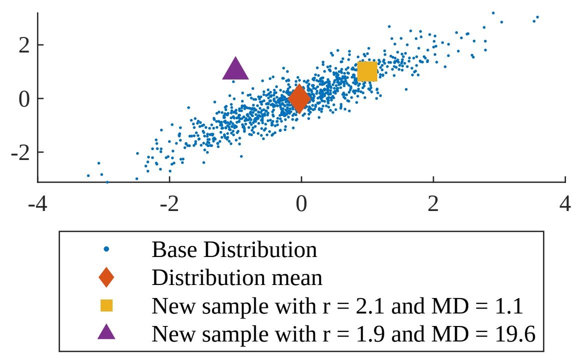


Figure 2.3: Mahalanobis distance.

Other measures

The peak value x_{peak} indicates the maximum value reached in a set of points. Moreover, the peak to peak value is a useful metric to indicate the range of distribution. It is defined as:

$$x_{peak2peak} = max(x) - min(x). \tag{2.11}$$

Finally, a useful feature for fault detection is the crest factor, which is the ratio of the peak level of a time series to its RMS value. This metric is used to detect changes in the signal pattern due to impulsive vibrations sources [79]. The crest factor is computed as :

$$Crest\ Factor = \frac{max(x)}{RMS}. \tag{2.12}$$

2.3.2 Signal preprocessing techniques

The preprocessing stage is a fundamental step before any classification or regression model is trained. Within this section, relevant feature preprocessing techniques for condition monitoring are introduced.

Feature transformation

The feature transformation includes manipulations that are applied to the original features. Through feature transformation it is possible to reduce the estimation errors or accelerate an algorithm prior to the transformation of the features [88]. The transformations considered within the frameworks that are proposed in this work are: centering, scale and standardization. To explain these transformations we first define f as a the vector containing the values of a feature for m samples with a mean over all samples $\overline{f}$ and a standard deviation σ_f.

Centering: centers the feature around zero, by setting $f_i = f_i - \overline{f}$

Scale: maps the values of the feature to the range $[a,b]$. It is computed as:

$$f_i = (b-a)\frac{f_i - f_{min}}{f_{max} - f_{min}} + a, \tag{2.13}$$

where f_{min} is the minimum value of the feature vector and f_{max} the maximum value.

Standardization: transforms f to a vector with zero mean and standard deviation of one. This metric is also known as Z-score normalization. It is given by:

$$f_i = \frac{f_i - \overline{f}}{\sigma_f}. \tag{2.14}$$

Noise reduction

Noise can be defined as “any unwanted disturbance that interferes with a desired signal” [16]. Most disturbances that affect sensor signals are caused by radiation from electrical equipment. The effect of these disturbances can be reduced with the implementation of a digital filter. For instance, a mean moving average. In sequential tasks, there should not be any leak of information from the future. Thus, the one sided moving average is applied. This is given by (2.15), where k is the number of observations used to compute the mean.

$$s_t = \frac{1}{k+1}\sum_{j=0}^{k} y_{t-j}, \ for \quad t \ = \ k+1, k+2, ..., n. \tag{2.15}$$

2.4 Dimensionality reduction

After the feature generation stage, the size of the data array is compressed, but the feature space can increase considerably. With the presence of a high dimensional space and a proportionately small number of samples, the learning algorithm tends to overfit, resulting on a performance decrease. Dimensionality reduction techniques for classification and regression problems are introduced in the following sub-sections.

2.4.1 Dimensionality reduction for classification

There are two main dimensionality reduction approaches for classification: feature extraction and feature selection. In feature extraction, the existing feature space is transformed into a new variable set, which can give better insight about the raw data. Then, only the most relevant features are selected to proceed with the training of the learning algorithm. One of the most known examples of feature extraction is the Principal Component Analysis (PCA). Even though the dimensionality is reduced with feature extraction methods, the complete original variable set has to be used to create these type of features. This means that all the raw variables have to be stored in a memory unit to then perform a prediction. However, in an embedded system this might result in an unfeasible task due to the lack of sufficient storage availability.

In feature selection, a subset of relevant features are chosen following determined evaluation criteria. This approach normally leads to a better performance, a lower computational cost and a better model interpretability [95]. Within the feature selection approach, there are three methods: filter feature selection, wrapper feature selection and embedded feature selection. The filter models do not require the utilization of a clasification algorithm. Instead, the features are ranked according to certain criteria and a threshold is defined to choose only those variables that exceed it. Most algorithms in this family are supervised, since they measure the relevance of a feature by its correlation with the class label [50]. An example of ranking methods are information gain, mutual information, mimimum redundancy maximum relevance and joint mutual information. These methods requiere low computational effort and most of them consider both the feature relevance and feature redundancy. A drawback of these models is the difficulty to find a suitable learning algorithm for the selected features, because the classification model is ignored during the selection procedure. Moreover, there is no ideal method for choosing the dimension of the feature space [14].

In the wrapper methods, the prediction performance of a given learning algorithm is used to evaluate a feature subset [25]. If the number of variables is not too large, an extensive search can be performed. However, the evaluation of all possible subsets requires 2^N operations, where N is the amount of features. The search can be accelerated by using heuristic search algorithms such as evolutionary algorithms. A main drawback of wrapper methods is the fact that classifiers are prone to overfitting, and thus the selected feature subset might achieve a great accuracy but with a low generalization power. This can be avoided by using a holdout test set to validate the generalization capability of the classifier with the selected feature subset. Finally, in the embedded approach, the features are ranked by importance with methods such as filter models. Afterwards, the lowest ranked features are sequentially eliminated and each feature subset is evaluated with a classification algoritm. This reduces the computation time compared to the normal wrapper method. Another method to perform the feature ranking is by using the weights of a classifier. In the following section the Support Vector Machine Recursive Feature Elimination (SVM-RFE) approach, which was proposed by Guyon et al. [26], is introduced.

Support Vector Machine Recursive Feature Elimination

The SVM-RFE approach performs a recursive elimination of the less relevant features until the desired performance has been achieved. The feature ranking is computed through the weight vector of the SVM as in (2.22), where a linear kernel function is used, such that the weight vector w is equivalent to $w = \sum_n \alpha_n \cdot y_n \cdot x_n$, where $\alpha \nleq 0$ for samples x_n that are support vectors, and y_n is the sample class. The resulting weight vector has a dimension of $1, N$, where N is the number of features. The ranking criteria c

for each feature n is given by $c_n = w_n^2$. Afterwards, the variable with the lowest ranking can be removed and a new feature subset is generated. The process is repeated until a target accuracy is achieved or after a defined number of iterations have elapsed.

2.4.2 Dimensionality reduction for regression

In the case of regression tasks, it is possible to evaluate the suitability of a variable as prognostic parameter by using the metrics introduced in [36]. These metrics are: monotonicity, trendability and prognosability; and are computed as follows:

- **Monotonicity** outlines the positive or negative trend of the feature. This metric is useful for systems which do not undergo self-healing during their lifecycle, such as bearings and gears. The monotonicity is given by:

$$Monotonicity = \frac{1}{M}\sum_{j=1}^{M}\left|\sum_{k=1}^{N_j-1}\frac{sgn(x_j(k+1)-x_j(k))}{N_j-1}\right|. \tag{2.16}$$

  where M is the number of systems observed, N is the number of observations per system.

- **Trendability** describes the similarity of a feature's trend among several measured systems. A feature with higher trendability has trajectories with the same underlying shape. It is obtained as:

$$Trendability = \min\left|corr\left(x_j, x_k\right)\right|, j,k = 1,...,M. \tag{2.17}$$

- **Prognosability** quantifies the variance on the last values of each system within a population. It is given by:

$$Prognosability = exp\left(-\frac{\sigma(x_j(N_j))}{\mu|x_j(N_j)-x_j(1)|}\right), \tag{2.18}$$

  where $x_j(N_j)$ is the last value of the measurement. A more prognosable feature has a lower variability in its end values, which makes the extrapolation of a feature to a failure more accurate.

The fitness function (2.19) of each prognostic parameter is calculated by the weighted average of the three metrics. The fitness enables the ranking of the variables. The weighting parameters w_m, w_t and w_p can be selected to give a metric more relevance in the fitness computation.

$$fitness = \frac{w_m\ monotonicity + w_t\ trendability + w_p\ prognosability}{3}. \tag{2.19}$$

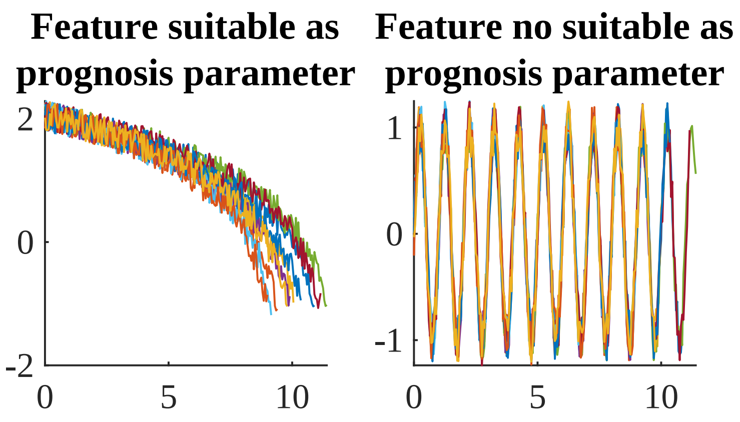


Figure 2.4: Example of a feature that is suitable as pronosis parameter and one variable that is not.

Figure 2.4 displays the examples of a good feature for prognosis and a variable that is not suitable as a prognostic parameter. The first one has a clear negative trend. Its monotonicity is equal to 0.81, the trendability to 0.99 and the prognosability to 0.96, which gives a total fitness of 0.92 with all the weighting parameters equal to one. This type of sensor signals are highly useful in the prediction of failures of a mechanical system. The second variable is a wave signal, whose behavior does not enable the forecasting of faults in mechanical elements. Its fitness is equivalent to 0.11, and due to its low ranking it would be immediately removed in the feature selection procedure.

2.5 Machine learning models for classification and regression

The traditional machine learning algorithms. which are described in the following subsections, use the information of the relevant preprocessed features to identify a fault type or to estimate the remaining useful life of a system.

2.5.1 Support Vector Machines

Support Vector Machine (SVM) is a supervised learning algorithm based on computational learning theory [12]. The key objective of SVMs is to construct a hyperplane $\mathcal{H}$ that separates two classes, so that the margin $\mathcal{M}$ is maximum between the hyperplane and the closest data points of the training set, which are called support vectors. A representation of this learning algorithm is depicted in Figure 2.5. The training set consists of N input vectors $x \in \Re^N$ with corresponding target values $y \in \{-1, 1\}$. The hyperplane that separates the two classes is given as:

$$\mathcal{H} = w^T \cdot \phi(x) + b = 0, \tag{2.20}$$

where $\phi(x)$ denotes a fixed feature-space transformation, $w \in \Re^N$ is a weight vector and b is the bias. The margin $\mathcal{M}$ has a width $2d = {}^{2}/_{||w||}$. The optimal solution requires to minimize $||w||$, which is equivalent to minimizing $||w^2||$. Therefore the problem can be transformed into a quadratic programming problem, which can be written as $argmin_{w,b}({}^{1}/_{2}||w^2||)$ subject to the constraints $y_i(w^T \cdot x_i + b) \geq 1$. This optimization problem is solved using the method of Lagrangian multipliers, hence:

$$L(w,b,\alpha) = \frac{1}{2}||w||^2 - \sum_{n=1}^{N} \alpha_n \cdot y_n(w^T \cdot \phi(x_n) + b) - 1, \tag{2.21}$$

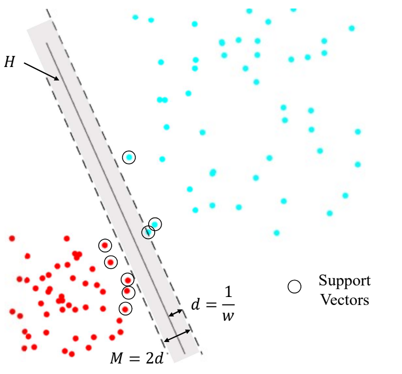


Figure 2.5: Representation of SVM.

where $\alpha \in \Re^N$ are the Lagrange multipliers. By setting the partial derivate of $L(w,b,\alpha)$ with respect to w equal to zero the solution for w is obtained as follows:

$$\frac{\partial L}{\partial w} = 0 \rightarrow w = \sum_{n=1}^{N} \alpha_n \cdot y_n \cdot \phi(x_n). \tag{2.22}$$

The weigth vector w depends only on the observations that are support vectors, which have nonzero coefficients α. By substituting (2.22) into (2.20), and $\phi(x)$ as the covariance function $k(x)$ we obtain:

$$y(x) = \sum_{n=1}^{N} \alpha_n \cdot y_n \cdot k(x, x_n) + b, \tag{2.23}$$

which gives a prediction for a new set of inputs. Common covariance functions $k(x)$ are displayed in Table 2.1

2.5.2 Gaussian Process Regression

A Gaussian Process (GP) is a group of random variables, for which any finite subset has a joint Gaussian distribution. The Gausian Process model has been widely used in classification and in regression tasks. A Gaussian Process Regression (GPR) model can make predictions incorporating prior knowledge and provides the uncertainty over predictions [9, 73]. A GP is defined by the mean function $m(x)$ and the covariance function $k(x_n, x_m)$ of a process $f(x_n)$, which are equivalent to:

$$m(x_n) = \mathbb{E}[f(x_n)], \tag{2.24}$$

$$k(x_n,x_m) = \mathbb{E}[(f(x_n)-m(x_n))(f(x_m)-m(x_m))], \tag{2.25}$$

where x_n and x_m are two random observations. The GP is given by:

$$f(x_n) \sim \mathcal{GP}(m(x_n),k(x_n,x_m)). \tag{2.26}$$

Figure 2.6 depictes an example of a Gaussian Process with its confidence interval

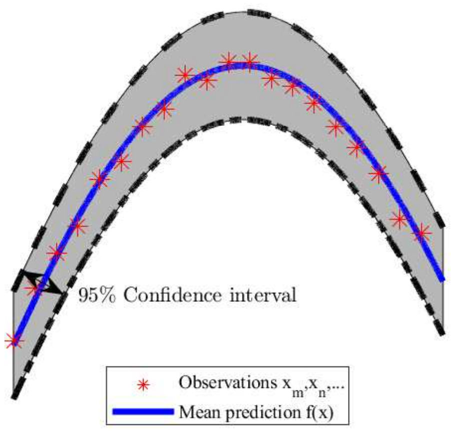


Figure 2.6: Representation of GPR.

Frequently used covariance functions $k(x_n,x_m)$ are summarized in Table 2.1, where σ_f is the signal standard deviation, σ_l is the length scale, α is a positive-valued scale-mixture parameter and r is the Euclidean distance given by:

$$r = \sqrt{(x_n - x_m)^T(x_n - x_m)}. \tag{2.27}$$

The coefficients σ_f, σ_l and α are known as the hyperparameters of the covariance function and can be together denoted by θ.

The hyperparameters σ_f, σ_l can be initialized by:

$$\begin{aligned}\sigma_l &= mean(std(x_{train})),\\ \sigma_f &= {}^{std(y_{train})}/\sqrt{2}.\end{aligned} \tag{2.28}$$

Table 2.1: Common covariance functions [72].

Covariance function $k(x_n,x_m)$	**Expression**
Polynomial	$(x_n \cdot x_m + c)^p$
Exponential	$\sigma_f^2 \cdot \exp\left(\frac{-r}{\sigma_l}\right)$
Squared exponential or RBF	$\sigma_f^2 \cdot \exp\left(\frac{-r^2}{2\sigma_l}\right)$
Matérn 3/2	$\sigma_f^2\left(1+\frac{\sqrt{3}r}{\sigma_l}\right)\exp\left(\frac{\sqrt{3}r}{\sigma_l}\right)$
Rational quadratic	$\left(1+\frac{r^2}{2\alpha\sigma_l^2}\right)^{-\alpha}$

The Bayesian linear regression model $f(x_n) = \phi(x_n)^T w$ is a common example of a Gaussian Process. Opposite to a linear regression model, the Bayesian model does not assume a linearity of the system and instead determines the posterior distribution for the model parameters. A Gaussian prior over the weights $p(w) \sim \mathcal{N}(0, \Sigma_p)$ is selected, so that the mean and covariance are:

$$\mathbb{E}[f(x_n)] = \phi(x_n)^T \mathbb{E}[w] = 0, \tag{2.29}$$

$$\begin{aligned} \mathbb{E}[f(x_n)f(x_m)] &= \phi(x_n)^T \mathbb{E}\left[ww^T\right] \phi(x_m) \\ &= \phi(x_n)^T \Sigma_p \phi(x_m). \end{aligned} \tag{2.30}$$

The joint distribution of the previous observations f and the function values f_*, according to the prior is:

$$\begin{bmatrix} f \\ f_* \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} K(X,X) & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) \end{bmatrix}\right), \tag{2.31}$$

where $K(X,X_*)$ denotes the matrix of the covariances of the previous and new observations. However, it is usually the case that we do not collect observations of the process itself but of a noisy version of it, therefore $y = f(x) + \varepsilon$, where ε is an additive Gaussian noise $\mathcal{N}(o, \sigma_n^2)$, so that $cov(y) = K(X,X) + \sigma_n^2 \cdot I$. By introducing the noise term into (2.31) we get to the joint distribution of the previous observations y and the function values f_* at the test location under the prior:

$$\begin{bmatrix} y \\ f_* \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} K(X,X) + \sigma_n^2 \cdot I & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) \end{bmatrix}\right). \tag{2.32}$$

The conditional distribution $p(f_*|X_*,X,f)$ is a multivariate normal distribution with mean and covariance given by:

$$m(X_*) = K(X_*,X)[K(X,X) + \sigma_n^2 \cdot I]^{-1} y, \tag{2.33}$$

$$\begin{aligned} cov(X_*) = K(X_*,X_*) - K(X_*,X) \\ [K(X,X) + \sigma_n^2 \cdot I]^{-1} K(X,X_*). \end{aligned} \tag{2.34}$$

Equations (2.33) and (2.34) can be rewritten in the following compact notation for a single new observation x_*:

$$m(x_*) = k_*^T (K + \sigma_n^2 \cdot I)^{-1} y, \tag{2.35}$$

$$cov(x_*) = k(x_*,x_*) - k_*^T (K + \sigma_n^2 \cdot I)^{-1} k_*, \tag{2.36}$$

where $K = K(X,X)$ and $k_* = k(x_*)$. The mean function (2.35) can be rewriten as:

$$m(x_*) = \sum_{i=1}^{n} \alpha_i \cdot k(x_i, x_*), \tag{2.37}$$

where $\alpha = (K + \sigma_n^2 \cdot I)^{-1} y$ and x_i is a previous observation. A new prediction is given by weighting every output with the similarity of its associated input to a new observation. Equation (2.37) depends only on the collected data and the covariance function, whose parameters are optimized by maximizing the log marginal likelihood given by [73]:

$$L = \log p(y|X,\theta) = \frac{1}{2} y^T (K + \sigma_n^2 \cdot I)^{-1} y - \frac{1}{2} \log |K + \sigma_n^2 \cdot I| - \frac{n}{2} \log 2 \cdot \pi. \tag{2.38}$$

The computation complexity in (2.38) is dominated by inverting the K matrix, which requires $O(n^3)$ operations for symmetric matrices, where n is the number of data points. For large datasets there are approximation methods, which reduce the computation complexity. Some of these approaches are: Subset of Datapoints (SD), Subset of Regressors (SR), Projected Process Approximation (PP) and the Bayesian Committee Machine (BCM). The computation complexity of each method is reported in Table 2.2. Here, m is a subset of datapoints from the original database, thus $m < n$. The selection of the optimal sample subset m can be achieved by greedy approximation. This algorithm together with the approximation methods introduced in Table 2.2 are explained in detail in [71]. Another approximation method that have demonstrated good efficiency are the Fully Independent Training Conditional (FITC) approximation and the Partially Independent Training Conditional (PITC) approximation, which were introduced in [69].

Table 2.2: Approximation methods for GPR with large datasets [71].

Method	Storage	Initialization	Mean	Variance
SD	$O(m^2)$	$O(m^3)$	$O(m)$	$O(m^2)$
SR	$O(mn)$	$O(m^2n)$	$O(m)$	$O(m^2)$
PP	$O(mn)$	$O(m^2n)$	$O(m)$	$O(m^2)$
BCM	$O(mn)$		$O(mn)$	$O(mn)$

2.5.3 Artificial Neural Networks

The term “Artificial Neural Network” (ANN) origined from the attempt to find mathematical representations of information processing in biological systems [10]. The most basic structure of an ANNs is the perceptron, which is composed of only one neuron. The schema of the perceptron is depicted in Figure 2.7.

Within the neuron, the vector of weighted inputs is summed with an offset value called

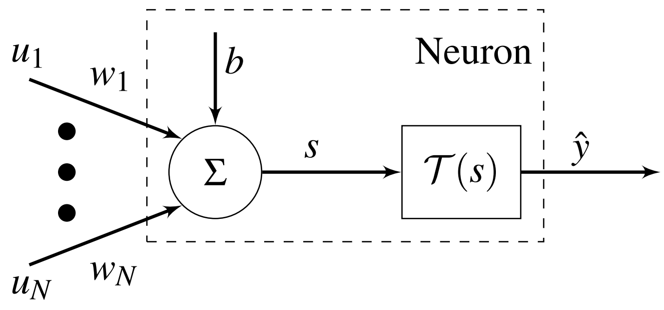


Figure 2.7: Schema of a perceptron.

bias. Mathematically, this process can be described as follows:

$$s = \sum_{i=1}^{N} w_i \cdot u_i + b, \tag{2.39}$$

where w_i refers to the weight i, u_i is the i-th input, b is the bias and s is the intermediate output. Then, the activation function $\mathcal{T}(s)$ maps the intermediate output of the neuron s to the scalar output $\hat{y}$ as follows:

$$\hat{y} = \mathcal{T}(s) = \mathcal{T}\left(\sum_{i=1}^{N} w_i \cdot u_i + b\right). \tag{2.40}$$

Commonly used activation functions and their derivatives are listed in Table 2.3.

Table 2.3: Activation functions $\mathcal{T}(s)$ [87].

Function	$\mathcal{T}(s)$	$\mathcal{T}'(s)$
Hyperbolic tangent	$\frac{1-e^{-2s}}{1e^{2s}}$	$1-\mathcal{T}^2(s)$
Sigmoid	$\frac{1}{1+e^{-s}}$	$\mathcal{T}(s)(1-\mathcal{T}(s))$
Linear	$c \cdot s$	c
ReLU	$\begin{cases} 0, & \text{if } s \leq 0 \\ s, & \text{if } s > 0 \end{cases}$	$\begin{cases} 0, & \text{if } s < 0 \\ 1, & \text{if } s > 0 \end{cases}$

Equation 2.40 can be represented in matrix form as:

$$\hat{y} = \mathcal{T}\left(\underline{w}^T\ \underline{u} + b\right), \tag{2.41}$$

where $\underline{w} = [w_1, w_2, ..., w_N]^T$ and $\underline{u} = [u_1, u_2, ..., u_N]$.

An extension of the perceptron is the feed-forward neural network. This is constructed by stacking more than one neuron into a layer, in which each neuron is connected to each weighted input. Moreover, if the network has several layers, then the architecture is known as the *multilayer perceptron* (MLP), which is a very succesful architecture in the context of pattern recognition [10]. The output $\hat{y}^{(l)}$ of the l-th layer is given by:

$$\hat{y}^{(l)} = \mathcal{T}^{(l)} \left(\underline{w}^{(l)} \cdot \underline{y}^{(l-1)} + b^{(l)} \right) \; for \quad l = 1, 2, ..., L \tag{2.42}$$

where $b^{(l)}$ is the bias vector with size $m^{(lx1)}$, $\hat{y}^{(l-1)}$ is the output of the previous layer with size $n^{(l)}$. The input of the first layer is the external input vector u, thus $y^0 = u$. The network's output is given by the outcome of the last layer $\hat{y} = y^{(L)}$. Finally, $\underline{w}^{(l)}$ is the weight matrix of the l-th layer and has a size $m^{(l)} x n^{(l)}$, where $m^{(l)}$ is the number of neurons of the l-th layer and $n^{(l)}$ is the number of outputs of the previous layer $l-1$. The amount of parameters of each fully connected (FC) layer is given by:

$$par_{FC} = m \cdot n + m. \tag{2.43}$$

The number of multiply-accumulate (MAC) operations of a FC layer are equivalent to:

$$.MAC_{FC} = m \cdot n \tag{2.44}$$

Equations (2.41-2.42) are used to describe the forward propagation of the neural network. The discrepancy between the network output and the desired output y, also known as loss function, is given by $e(\underline{w}) = (y - \hat{y}(\underline{w}))$. This loss function can take the following quadratic form:

$$E(\underline{w}) = \frac{1}{M} \sum_{i=1}^{M} (y_M - \hat{y}_M(\underline{w}))^2, \tag{2.45}$$

where M is the number of outputs. This function is known as the mean squared error loss function, which is used for regression problems.

The training of the network can take place by gradient descent algorithms, which use the gradient $\nabla E(\underline{w})$ of the quadratic error $E(\underline{w})$ to perform the learning [86]. The back-propagation algorithm, proposed by [76], is the most known approach to compute this gradient.

2.6 Deep Learning Models for Regression

2.6.1 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are a type of Neural Networks (NN) that use convolutions in place of general matrix multiplication in at least one of their layers [22]. CNNs are specialized to process data that has a known grid-like topology, for example time series and image data [22]. In a traditional NN, all the inputs are connected to all the neurons in the input layer, which generates an exponential increase of the number of parameters when inputs such as 2D images are given. On the other hand, the use of convolutions reduces drastically the number of total parameters of the network. The convolution operation is given as:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{a=\infty} x(a)w(t-a) \tag{2.46}$$

where w is the kernel and $*$ represents the convolution operation. In one dimensional (1D) CNNs, equation (2.42) of a MLP is substituted by:

$$\hat{y}^{(l)} = \mathcal{T}^{(l)}\left(\underline{y}^{(l)-1} * \underline{K}^{(l)} + b^{(l)}\right) for\ l = 1,2,...,L, \tag{2.47}$$

where $\hat{y}^{(l-1)}$ is the output of the previous layer with size $\{H,W,C\}$. Here H is the height, W the width and C the number of channels. In 1D CNNs, the feature map has a height of 1. Thus, the size of the input map is given by $\{W,C\}$. The width refers to the time steps and C to the number of sensor signals. Moreover, $\underline{K^{(l)}} = \left[K_1^{(l)}, K_2^{(l)}, ..., K_k^{(l)}\right]^T$, where k is the number of filters and each $K_i^{(l)}$ is the $i-$th 1D kernel for the layer l with size k_s. The bias b is added to the convolution operation of each filter to obtain the prior output s^l. The output of the layer l is given by the activation function $\mathcal{T}$. CNNs usually use a Rectified Linear Unit (ReLU) [64] activation function, which is given by:

$$\mathcal{T} = \begin{cases} 0, & \text{if } s \leq 0 \\ s, & \text{if } s > 0 \end{cases} \tag{2.48}$$

The size of the resulting feature map is given by:

$$\begin{aligned} W_{out} &= \frac{W_{in} - k_s + 2 \cdot P}{S} + 1, \\ C_{out} &= k, \end{aligned} \tag{2.49}$$

where S is the stride, which controls how the filter convolves around the input image, and P is the padding. Figure 2.8 depicts the 1D convolution operation between a feature map and the filters of a 1D CNN layer.

The number of parameters of a convolutional layer is given by:

$$par_{CN} = k(k_s \cdot C_{in} + 1). \tag{2.50}$$

Moreover, the number of MAC operations of a convolutional layer with $W_{in} = W_{out}$ is equivalent to:

$$MAC_{CN} = k_s \cdot C_{in} \cdot C_{out} \cdot W. \tag{2.51}$$

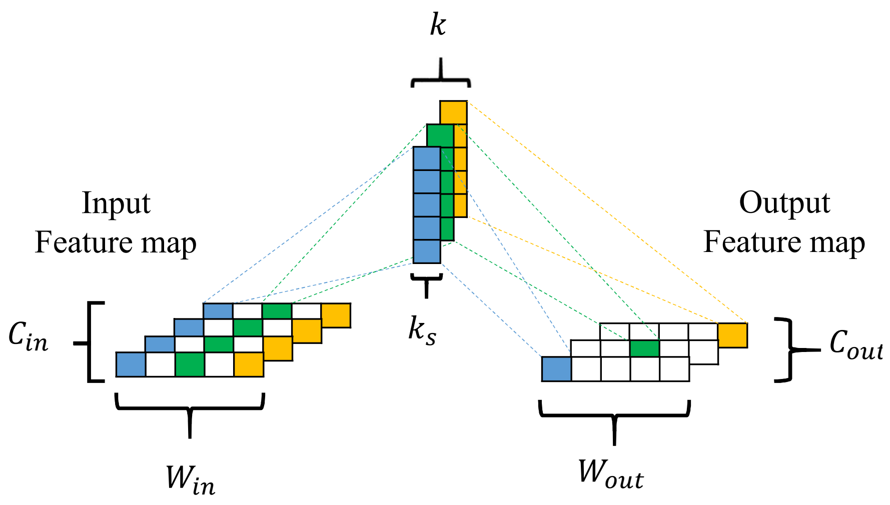


Figure 2.8: 1D convolutional layer. © 2020 IEEE [62].

In deep neural networks, the dimensionality reduction of feature maps is performed by adding a *pooling* layer. This layer reduces the dimension of the feature map by a factor given by the *pooling stride* p_s. For example, a pooling layer with $p_s = 2$ reduces each $\{2,2\}$ segment of the feature map to only one element. There are two usual methods to define how the reduced element is obtained. The first one is known as *max pooling*, where the maximum value among the segement is chosen. The second one, the *average pooling*, outputs the mean value of the elements.

A method to reduce the computational complexity of CNNs is by using depthwise separable convolutions (DSC) instead of standard convolutions. The DSCs factorize a standard convolution into a depthwise convolution and a pointwise convolution [32]. The DSC applies a single filter for each channel of the input ferature map. The pointwise convolution is used to combine the output of the depthwise convolution with a 1x1 convolution to generate a new feature map. The MAC operations of a layer with depthwise separable convolutions is given by [19]:

$$MAC_{DSCN} = W \cdot C(ks + C_{out}). \tag{2.52}$$

2.6.2 Temporal Convolutional Networks

To improve the efficiency of CNNs for sequence modeling, Bai et al. [6] introduced an architecture belonging to the Temporal Convolutional Network family, and for simplicity they labeled it with the same name. Their architecture has three main features: the ability to map an input of any length into an output of the same length, which is performed with a zero padding, the use of causal covolutions to avoid the leakage of information from future to past, and the possibility of the network to look very far into the past to perform a prediction by using dilated convolutions.

Causal and dilated convolutions

During the deployment of an algorithm for sequential modeling, the outcome should depend only on the information from the current and previous steps, such that the prediction $p(x_{t+1}|x_0,...,x_t)$ has no dependencies on future timesteps $x_{t+1}, x_{t+2},...,x_T$, which is achieved by using causal convolutions. Models with causal convolutions are faster to train than RNNs, due to the lack of recurrent connections. However, they require a higher number of layers or a larger kernel size to increase the receptive field [1]. This complicates the use of causal convolutions for sequence tasks that require a long history. On the other hand, the TCN uses dilated convolutions to exponentially increase the receptive field with a minor increase in the network size. The dilated convolution F at the step s of a sequence is defined as:

$$F(s) = (x * d_f)(s) = \sum_{i=0}^{k_s-1} f(i) x_{s-d_i}, \tag{2.53}$$

where d is the dilation rate, x is the 1D sequence input, f is a filter with size k_s and $s-d_i$ accounts for the direction of the past.

The number of elements r that a single filter can visualize at each level of the network is given by $r_j = 1+(k_s-1)2^{d_j}$, where d_j is the dilation rate at the level j. The dilation rate is exponentially increased with the depth of the network, such that $d = O(2^j)$ at level j of the network. Moreover, the total receptive field RF that can be covered by a TCN with a constant kernel size in all its layers is given by:

$$RF = 1+(k_s-1)\sum_{j=0}^{d^N} 2^j, \tag{2.54}$$

where d^N is the dilation rate at the top level.

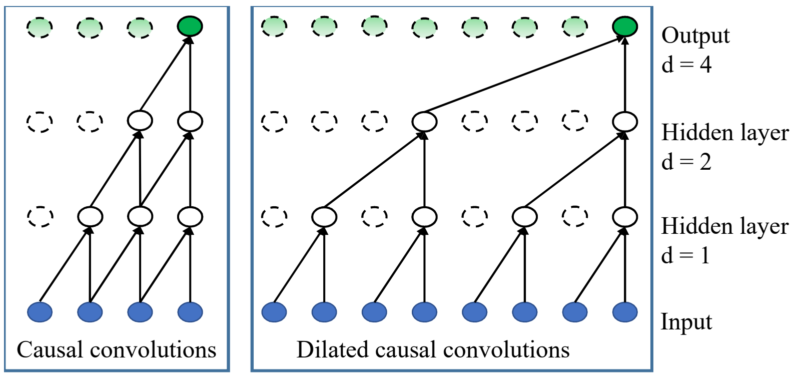


Figure 2.9: Visualization of causal and dilated causal convolutions. © 2020 IEEE [62].

Fig. 2.9 shows the visualization of two networks, one with causal convolutions and the other with dilated causal convolutions. In both architectures, the filter size is equal to two. The model with causal convolutions has a receptive field of four. Meanwhile, when dilated convolutions are introduced, the receptive field is increased to eight without increasing the number of layers or kernel size.

Residual block

With an increased number of hidden layers, a network can face a *degradation* problem, where both the training and validation accuracy get saturated and then degrade rapidly [27], contrary to overfitting, where only the validation accuracy worsens. To cope with this problem, authors in [28] introduced the residual blocks, where the output of a series of transformations F are added to the input x of the block. The residual block proposed by [6] consists of two layers of dilated causal convolutions, whose weights are normalized with a weight normalization [81] and activated by a ReLU [64]. Moreover, a spatial dropout [99] is used after each activation function for regularization. Finally, the output of the second convolutional layer is added with the input of the block, which is connected to a 1D convolution to ensure an uniform tensor shape in the addition. The architecture of the residual block is depicted in Fig. 2.10.

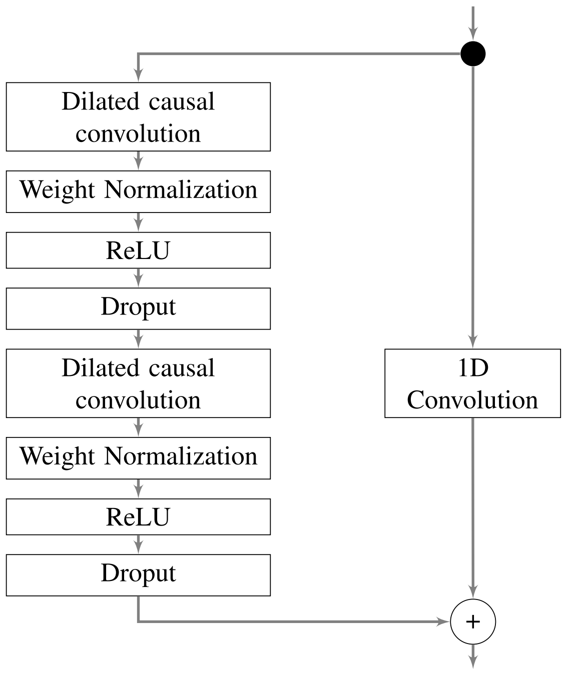


Figure 2.10: Residual block. © 2020 IEEE [62].

2.7 Regularization methods

A common problem that is faced when training both machine learning and deep learning models is overfitting. “A model overfits the training data when it describes features that arise from noise or variance in the data, rather than the underlying distribution from which the data were drawn” [108]. Overfitting is directly related to the loss of performance for validation and test data, since the model is not able to generalize. Regularization methods have the objective to reduce overfitting and thus, increase the generalization capability of the model. One common regularization technique is the dropout [94], in which the activation of neurons are zeroed with a probability p_{drop}. This prevents that the activations become strongly correlated. Furthermore, authors in [99] proposed the Spatial Dropout, which "drops out" full feature maps instead of single neurons. Batch normalization [34] is another regularization method, which normalizes by means of the z-score standardization the set of activations in a layer.

Other regularization methods such as data augmentation improve the models performance with unseen samples but also are useful to train learning algorithms with reduced available data. The objective of data augmentation is to generate new inputs from the existing training database. Thus, deep learning models can be optimized even with a low amount of available data. The idea behind data augmentation is that more information can be extracted from the original database by augmentations [91]. Basic time domain methods for data agumentation are: window cropping, window warping, flipping, noise injection [109]. The window cropping or window slicing method randomly extracts continuous slices from the original time series [17]. Each generated slice mantains its original label. In window warping, a segment of the time series is randomly selected and then compressed or extended [109]. This technique changes the size of the window. Thus, the windows that are compressed should be generated from feature maps larger than the size of the inputs to the learning models. On the other hand, if the windows are extended, the original size should be lower to that of the feature map. In this way, a constant feature map size can be assured. Flipping is a technique where a new time series is generated by flipping the sign of the original time series. Finally, noise injection consists on injecting a vector of random values, usually drawn from a Gaussian distribution [91].

2.8 Optimization methods

In machine learning and deep learning, an optimal model can be obtained by trial and error. However, this task is very time consuming and could be unfeasible for a high dimensional hyperparameter space. To automate this process, optimization methods are integrated into the search of optimized hyperparameters. Within this section, the optimization methods of Simulated Annealing and Genetic Algorithm are introduced.

2.8.1 Simulated Annealing

Simulated Annealing (SA) is an optimization method, which is based on the metallurgical process of annealing [98]. Annealing is a heat treatment of material with the objective of altering its properties such as hardness. The goal of SA is to minimize the value of the energy (E), which quantifies the performance of a determined system. The first step of SA is to compute the energy of the system without any modification. Afterwards, the properties of the system are changed and the new energy (E_{i+q}) is obtained. Then, the change in energy (ΔE) is calculated by (2.55) and the probability $P(\Delta E)$ that the modified system is accepted is given by (2.56), where T is the temperature.

$$\Delta E = E_{i+1} - E_i \tag{2.55}$$

$$\mathcal{P}(\Delta E) = \begin{cases} e^{-\Delta E/T} & \text{for } \Delta E > 0 \\ 1 & \text{otherwise} \end{cases} \tag{2.56}$$

If the modified system has a lower energy, the transition is accepted. Otherwise, a distributed random number $r\ \varepsilon[0,1)$ is drawn and the step will be accepted only if $r < P(\Delta E)$. With high temprature values, the probability $P(\Delta E)$ is close to 1, leading to the aceptance of many uphill steps. On the other hand, low temperatures hinder that the system escapes local regions and generally, only transitions with lower energy values than the current lowest energy value are accepted. Therefore, at the beginning of the search, high temperature values are selected to explore diverse solutions. Towards the end of the search, the temperature drops to values close to zero. The new temperature value for each accepted transition is given by $T_{i+1} = T_i \cdot \Delta T$.

2.8.2 Genetic Algorithm

A genetic algorithm (GA) is an optimization method, which is based on the biological concept of evolution. The GA has five main steps: initialization, evaluation, selection, crossover and mutation [44]. Figure 2.11 depictes the representation of how these steps iterate to achieve an optimal solution.

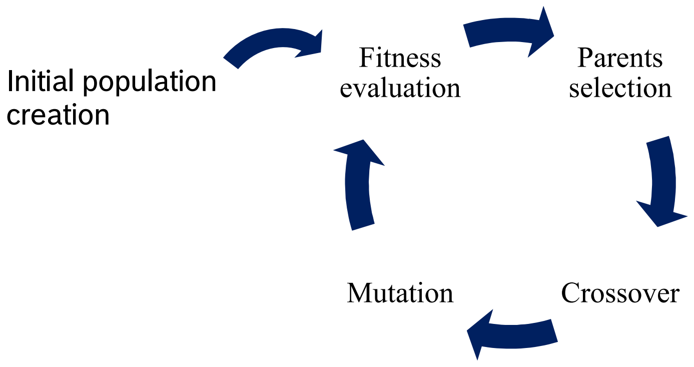


Figure 2.11: Representation of a GA [44].

First, an initial population is created. This population has a size of $\{m,n\}$, where m is the amount of individuals, which build the initial population, and n is the number of genes of each individual. The genes are the parameters to be optimized. The initialization should randomly cover the whole solution space, or be based on expert knowledge [44]. Afterwards, each individual is evaluated by a fitness function. This function should be established such that lower values represent a better solution. In this way, we assure that the search is a minimization problem. Once the fitness of each individual is obtained, a selection of the so-called *parents* takes place. There are several methods to select the parents, such as: truncation, fitness proportionate, tournament, linear ranking and random selection. These approaches are described in detail in [96].

The crossover is the stage at which the genetic material of two or more solutions is combined to generate offsprings [44]. This includes two steps, which are the mating of the parents and the crossover of their genes. Since the number of selected parents is normally greater than two, it is neccesary to define which individuals mate with each other. One option is to mate the best parent with the worst one, the second best with the second worst and repeat this successively until all individuals are mated. Another method is to perform a random mating of the parents. Afterwards, the genes that are exchanged between the pairs are defined. One common approach is the single-point crossover, where both parent genes are split at a randomly determined *crossover point* [97]. Then, the first offspring is created with the first part of the genes of the first parent and the second part of the genes of the second parent. Subsequently, a second offspring can be generated with the second part of the genes of the first parent and the first part of the genes of the second parent. Other typical methods to perform crossover are the two-point and multi-point crossover, as well as the uniform crossover [97]. It is as well possible to perform a random selection of the crossover genes.

The selected parents and the generated offspring constitute the new population. In order to analyze more possible solutions, a percentage of the genes is mutated with a fixed or variable rate known as the *mutation rate*. The search concludes when a termination condition has been fulfilled. Possible termination conditions are when a predefined amount of iterations or a desired fitness value has been reached.

Chapter 3

State-of-the-Art

In the first part of this chapter, well-established intelligent algorithms that enable the diagnosis of the fault type and the prediction of the system's remaining useful life are introduced. In the second part of this chapter, a summary of the criteria that are fulfilled by the approaches previously described within the chapter is given. This enables the identification of the gap in the literature, which is filled with the development of the present work.

3.1 Fault diagnosis approaches

The intelligent fault diagnosis (FD) methods can be separated in traditional machine learning (ML) and deep learning (DL) techniques.

3.1.1 Traditional machine learning methods for FD

The machine learning methods for fault diagnosis follow a very similar sequence of steps. This includes the extraction of time-, frequency- or time-frequency-based features; the selection of the most relevant features and the training of the classification algorithm. Some approaches focus on diagnose if there is a fault present on the system, while others focus in the fault isolation.

The SVM has been implemented in several works to diagnose the fault type. In [78], authors collected data from an experimental setup that tested different bearing conditions. After the feature extraction, the dimensionality reduction was performed with a decision tree. The selected features were used to train an SVM. Four different covariance functions were tested and the radial basis function (RBF) delivered the highest accuracy. In [43], the feature extraction was performed with the wavelet transform (WT). Then, a SVM was trained with the extracted wavelets, which enabled an accurate fault identification. The generalization capability of the SVM can be increased by using a transductive support vector machine (TSVM). The TSVM uses labeled and unlabeled data during the training. In [90], authors implemented an TSVM to accurately diagnose gear faults. SVMs have also demonstrated a good performance in the diagnosis of faults caused by mechanical elements of electric motor, besides gears and bearings, as described in [51].

The ANNs have been widely used in fault diagnosis as well. The MLP was implemented in [70] to identify damaged gears and bearings of a gearbox. In [47], a RBF network served as the classification algorithm, which diagnosed gear faults. Besides accelerometer signals, motor phase currents provided relevant information for the isolation of a malfunction. In [5], authors used the motor currents to train an ANN to diagnose faults in an induction motor. Another useful ML model for fault diagnosis is the k-nearest neighbors (k-NN). Authors in [105], developed an approach based on k-NNs to identify different gear crack levels under varying operation settings.

Two common aspects of fault diagnosis approaches based on traditional machine learning models were identified. First, a test bench was built in each work to generate the data. Most authors used accelerometers and sampled them with high rates, which enabled the extraction of frequency- and time-frequency-based features. This brings us to the second common aspect, which indicates that authors of the introduced approaches trained their models with the extracted wavelets, which demonstrated to be very useful to identify the fault types. Another works that present methods for fault diagnosis of rotating machinery with machine learning models are summarized in [55].

3.1.2 Approaches based on deep learning architectures for fault diagnois

DL architectures have shown a considerable increase on performance compared to traditional ML models. CNNs are the most commonly used DL model for grouping tasks such as image classification. They can be easily adapted to classify faults in mechanical elements of rotating machinery. The convolutional layers are able to extract features directly from raw data. Thus, a previous manual feature engineering is not mandatory. However, a feature extraction is very helpful when dealing with signals that were collected with very high sampling rates, which is the case of accelerometers signals.

In [24], the continuous wavelet transform (CWT) was computed from accelerometer and displacement signals. The CWT was used to build feature maps, which were used to train an architecture with 1D convolutional layers and a FC layer to diagnose among several fault types in a rotating machine. Authors in [114], computed the Hilbert-Huang transform from vibration signals. The Hilbert spectrums were used to construct 2D images that were given as input to an CNN that could diagnose three fault types of ball bearings. Moreover, Zhang et al. [115] proposed a method named CNN with training interference (TICNN). This architecture could process raw vibration signals directly without the need to perform a feature extraction. The framework introduced in [115] is depicted in Figure 3.1. The multiscale CNN [39], is another approach that can directly extract features from raw vibration signals. This novel approached enabled the diagnosis of different fault types that arose in a wind turbine gearbox.

The use of continuous vibration signals make the fault diagnosis a sequential task, which can be solved by using long short-term memory (LSTM) networks. Proposed by

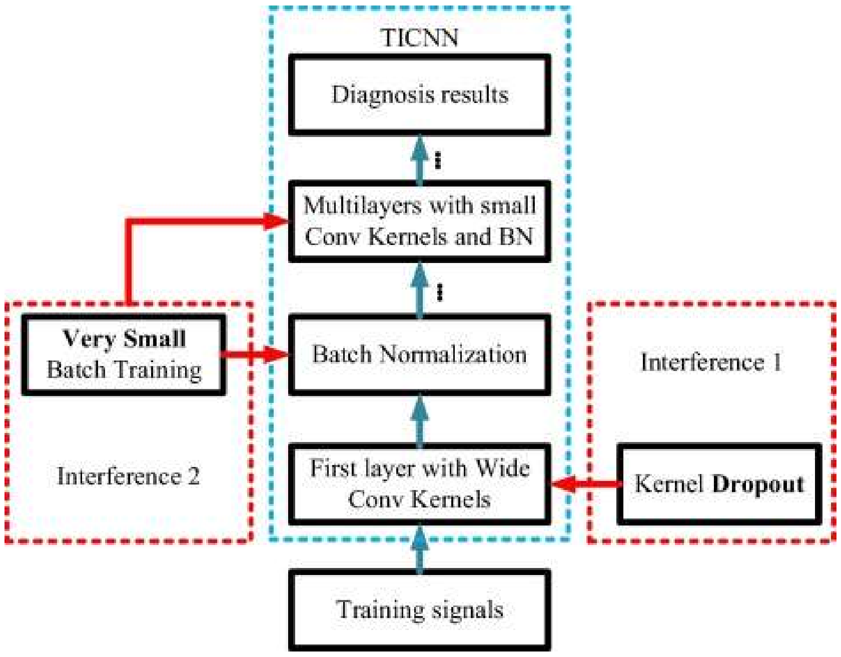

TICNN
Diagnosis results
Multilayers with small Conv Kernels and BN
Very Small Batch Training
Batch Normalization
Interference 1
Interference 2
First layer with Wide Conv Kernels
Kernel Dropout
Training signals

Figure 3.1: Framework of TICNN[115].

Hochreiter et al. [31], this type of network is able to cope with the problematic of exploding and vanishing gradient problems faced by traditional Recurrent Neural Networks (RNN). In [117], a DL architecture with LSTM layers extracted features from raw signals to isolate a malfunction within 21 fault types that were generated with a simulation framework, which reproduced industrial processes. Moreover, authors in [112] proposed a network with three LSTM layers, which was able to correctly diagnose bearing faults. In [66], a novel architecture that consists of CNN and LSTM layers was presented. This model demonstrated a higher performance to diagnose bearing faults than models that only used CNN or LSTM layers.

DL architectures require a great amount of data to converge. Many authors of the presented approaches built experimental setups to collect the data or used open source databases. Moreover, other authors used deep networks previously trained for image classification and retrained their last layers to perform the FD. This technique is known as transfer learning. In [13], authors used the pre-trained deep CNN network presented by Krizhevsky et al. [45], which was used to classify images from the big dataset *ImageNet [18]*. The network consisted of five convolutional layers and three dense layers and it was originally created to sort an image out of 1000 different classes. Within the approach in [13], only the last layer was retrained using the experimental data, which consists of vibration signals collected from a benchmark gearbox. The raw signals were resized to grayscale images by using the bicubic interpolation. Similarly, Shao et al. [89] retrained the deep network VGG-16 [93], which was trained with the *ImageNet* dataset as well, to identify the condition of an electric rotating machine. They used the CWT to transform accelerometer signals into the time-frequency domain. Then, a channel augmentation was performed to obtain 2D images with three channels, which served as input to the CNN network. Transfer learning based approaches demonstrated a very high performance to diagnose faults of mechanical elements even with a low amount of data available.

3.2 Remaining useful life estimation techniques

The remaining useful life (RUL) can be defined as the period between the current time step and the end of the product's useful life [92]. In the literature, there are two main methods for health prognosis: the physics model-based approach and data driven techniques. The first method requires a deep knowledge on the component's physical processes. Thus, an analytical model, which describes the behavior of the elements within the system and their interactions, can be built. This can be a highly complex task for certain products, such as machinery, aircraft engines or automotive components. Therefore, around 90% of current works focus their research on the development of data-based approaches for RUL prediction [48]. These techniques make use of statistical methods or artificial intelligence approaches. Within this section, we focus on two types of algorithms for RUL estimation: the traditional machine learning approaches and deep learning architectures. The generation of enough data that enables the creation of reliable prognosis models is a highly complex task. Thus, algorithms for RUL estimation are validated mainly with publicly available datasets like C-MAPSS [83] and PRONOSTIA [65].

3.2.1 Traditional machine learning algorithms for RUL prediction

The approaches that implemented traditional machine learning models had a common sequence of steps, which included the feature generation, feature selection, preprocessing, health index creation and training of a regression algorithm. First, time-based or frequency-based variables were extracted from raw signals, depending on the sample rate, with which the data was collected. Then, dimensionality reduction approaches were used to select the variables that were more suitable as prognosis parameters. After the signal preprocessing, some approaches required the computation of a health index (HI), which assessed the condition of the system within its lifetime. Finally, either the computed HIs or the selected features were given as the input vector or matrix to the regression algorithm. It is at the last step, where the proposed approaches mainly differentiated from each other by using different machine learning models.

Feed forward ANNs have been implemented to efficiently forecast the RUL of rotating systems. In [7], authors proposed an approach to predict the RUL of turbofan engines by training an ANN with the raw sensor signals and an HI. Moreover, ANNs have also been implemented to accurately predict the failure time of ball bearings, which were monitored with accelerometers with high sampling rates ($> 20kHz$). Authors in [57] generated time-based features from raw vibration signals. With these signals a Weibull hazard rate function was fitted to generate an HI. The time and HIs were used as inputs to train an ANN, which delivered the percentage of the bearing's life. Similarly, a hazard rate function was computed from time based features of accelerometer signals in [107]. The extracted HI served as input to an ANN, which delivered the survival probability of the test bearing. Furthermore, RNNs are an extension of ANNs that are commonly used for sequential modeling tasks. In these networks, the predictions of $n-$ steps in the past are given as input to predict the value of the step $k+1$. Authors in [29] implemented this architecture to forecast the RUL of turbofan engines.

SVMs for regression, also known as support vector regression (SVR), have shown a great accuracy in failure prognosis of mechanical elements of different systems. In [42], the failure forecasting of aircraft engines was performed with a SVR that was trained directly with sensor signals, thus avoiding the need to compute an HI. Moreover, authors in [8] proposed an approach to estimate the RUL of rolling bearings with SVR. This model was trained with HIs that were computed by means of the isometric feature mapping reduction technique. Finally, Saidi et al. [77] developed a methodology to predict the time to failure (TTF) of rolling bearings of wind turbines. The novelty of the approach is that the generated time-based features were not directly used as input to the model, but the spectral kurtosis of each feature were used to train the SVR, which demonstrated an increase on performance of the model. Another kernel method that has been implemented for failure forecasting is the Gaussian process for regression (GPR) model. In [13], authors developed a method to predict the TTF of wind turbine bearings. First, they computed the WT from raw sensor signals. Then, the interval whitenization method was implemented to reduce the noise in the wavelets, which was caused by the non-stationary operation conditions of the evaluated system. Finally, the GPR model constructed the relation between the wavelets and the RUL.

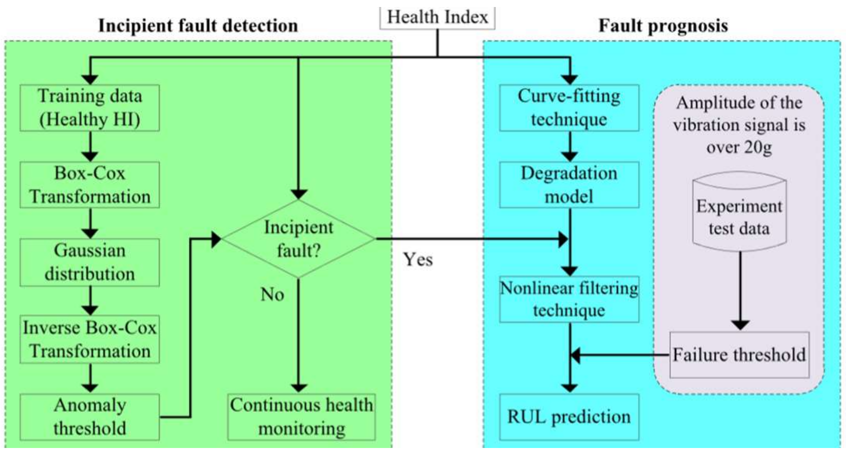


Figure 3.2: Anomaly detection and fault prognosis approach. © 2016 IEEE [40].

The aforementioned approaches covered only the failure forecasting. However, none of them described a method to determine a threshold value that indicated the arise of an anomaly. Contrary to these approaches, authors in [40], proposed a methodology that covered both the fault identification as well as the failure prognosis. Their approach is depicted in Figure 3.2. The fault detection was performed by computing an HI with the Mahalanobis distance. Then, the Box-Cox transformation was implemented to normalize the data and be able to obtain the anomaly threshold. The limit values were returned to the original distribution with the inverse Bos-Cox transformation. Once an anomaly was detected the RUL estimation took place by means of an extended Kalman filter.

3.2.2 Deep learning architectures for FP

DL architectures are composed of several layers with hundreds, thousands or even millions of parameters, which enables an automatic identification of degradation patterns directly from the raw data. Thus, several of the previously described steps neccesary for the deployment of a ML model for RUL estimation can be dismissed with the use of DL architectures. Only the data normalization remains as an essential step prior to the training of the deep learning models. For instance, LSTMs have been widely used for RUL estimation. In [110, 113, 118], LSTM layers were included into deep architectures to predict the RUL of turbofan engines achieving a greater accuracy than models that make use of RNNs [29]. Their architectures are very similar, and the only distintion between these methods is the number of LSTM cells that are used in each layer of the architecture. The best performance within these models was obtained by authors in [118]. Their architecture consisted of 4 hidden layers with 32 nodes in the first two layers and 8 nodes in the last two.

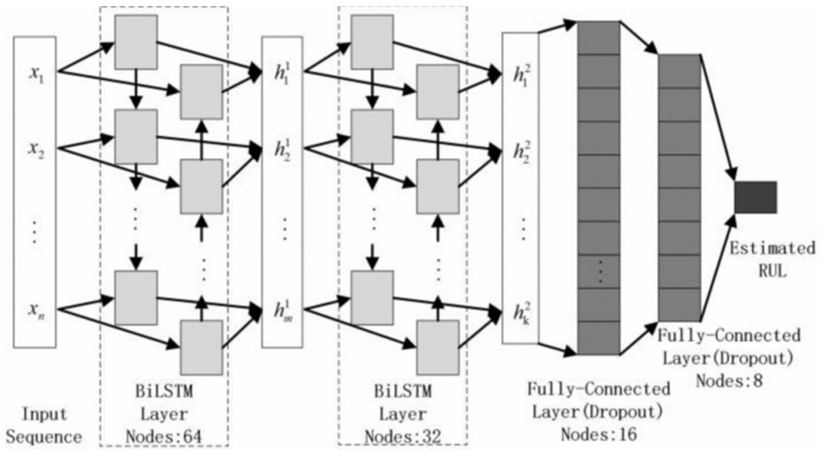


Figure 3.3: BiLSTM architecture for RUL estimation. © 2018 IEEE [106].

In [106], authors built a deep architecture with two BiLSTM layers and two fully connected layers to predict the RUL of aircraft engines. In Figure 3.3, this BiLSTM architecture is displayed. Huang et al. [33] extended this BiLSTM architecture by adding a third BiLSTM layer. The input of this additional layer was the output of the second BiLSTM and the operating conditions. This layer was then connected to a FC layer, which delivered the estimated RUL of turbofan engines. This architecture improved the performance of the approach described in [106] by almost 10% in the most complex subset of the C-MAPPS [83] database. Moreover, authors in [116] proposed an architecture for RUL estimation of turbofan engines based on LSTMs and named it as LSTM-Fusion Network. This network first passed the sensor signals with variable window size through a series of LSTM sub-networks, which were independent from each other. Each sub-network consisted of two LSTM layers. Afterwards, the outputs of the sub-networks were concatenated in the LSTM-Fusion layers. Finally, the fusion layers were connected to a dense layer that provided the estimated RUL. This architecture demonstrated a great performance in the C-MAPPS dataset.

Authors in [4] proposed a semi-supervised architecture based on LSTMs to estimate the TTF of aircraft engines. First, an unsupervised pre-training stage with a restricted Bolzmann machine took place. This helped to extract degradation related features from raw unlabeled data. The extracted features were used as input to a network with two LSTM layers and a fully connected layer. The hyperparameters of the network were optimized with a genetic algorithm.

One problematic that arises in the embedded deployment of LSTMs is their high computational complexity, which for a single LSTM layer is given by:

$$MAC_{LSTM} = 4(n \cdot m + n^2 + n) \cdot W, \tag{3.1}$$

where m is the input size or channel number, n is the size of the output of the LSTM layer, which is equivalent to the number of hidden units, and W is the width of the feature map. Moreover, the number of parameters of an LSTM layer is equal to C_{LSTM}/W. From these equations, we can identify that the number of hidden units has the greatest influence in the size and computational complexity of an LSTM layer. Furthermore, the computational complexity of a BiLSTM layer is given by: $MAC_{BiLSTM} = 2 \cdot MAC_{LSTM}$.

CNNs, widely used for image classification, can perform predictions with continuous data by substituting the 2D and 3D convolutional layers (CL) with 1-D CLs. Authors in [82, 52] proposed a 1D CNN architecture with two and four CLs respectively to estimate the RUL of aircraft engines. Due to the low amount of samples for each turbine, a prior feature extraction was not required. On the other hand, when the condition of a system was monitored with sensors with high sampling frequencies, a feature extraction stage was performed by most authors. Zhu et al. [120] computed the WT of accelerometer signals and used these variables to train a multiscale CNN to estimate the TTF of rolling bearings. Their proposed approach and the multiscale CNN are depicted in Figure 3.4. Authors in [74] computed the frequency spectrum of raw accelerometer signals with the discrete Fourier transformation and constructed from this spectrum the feature maps, which were used to train a deep CNN for RUL estimation of bearings.

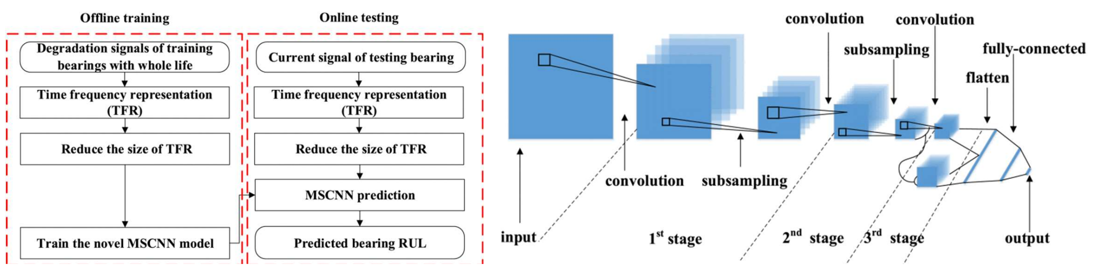


Figure 3.4: Multiscale Convolutional Neural Network. © 2019 IEEE [120].

In [104], authors proposed the use of DSCs in their deep architecture for RUL estimation instead of standard convolutions. They implemented the DSCs within the residual blocks that constituted their network. The residual block also had a so-called squeeze and excitation unit after the DSCs. The functionality of this unit was to highlight informative feature maps and supress useless ones. The novelty of their deep architecture was that it could process signals that were sampled with high frequencies without the need of a manual feature extraction. Moreover, the use of DSCs reduced the computation complexity. The deep architecture proposed in [104] is depicted in Figure 3.5.

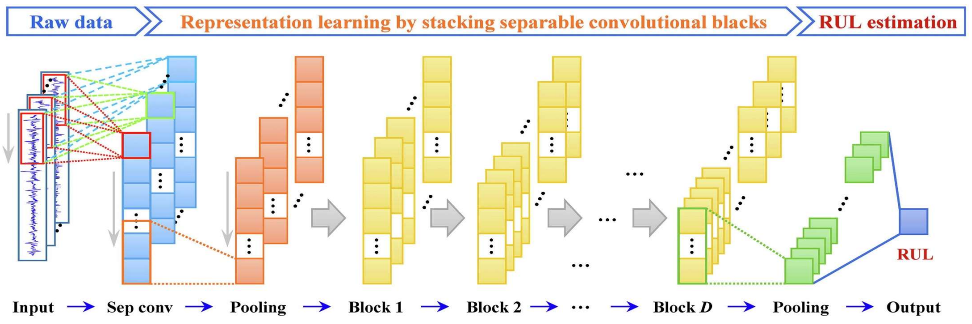


Figure 3.5: Depthwise Separable Convolutional Network for RUL estimation [104].

The receptive field (RF) of a CNN, which is the region that a convolutional feature covers, can be exponentially increased with only a linear increment of the parameter number by using dilated convolutions. In machinery health prognosis, the increase of the RF allows the network to gain more knowledge about the machine condition in the past with a minor increase in the network's size. Dilated convolutions can increase the performance of a prognostics model based on normal CNNs as shown in [111]. In [54], authors presented a method to forecast the RUL of rolling bearings based on TCNs, achieving a better perfromance than LSTM models with a *lighter* network that required much less training time than such approaches. Moreover, Zhou et al. [119] demonstrated the functionality of TCNs for the RUL estimation of Lithium-Ion batteries. Finally, an architecture with TCN and LSTM layers was proposed in [38] to estimate the RUL of aircraft engines.

3.3 Fulfilled criteria by existing state-of-the-art approaches

Table 3.1 presents 16 related approaches for condition monitoring. The first seven methods are based on traditional machine learning models. The last nine methods use a deep learning architecture for the classification or regression task. Each column of the table represents a requirement of the comprehensive methodology. If an approach fulfills a criterion, the cell is highlighted in green.

Regarding the performance, both ML and DL models have demonstrated high accuracy for fault diagnosis. As for the failure prognosis, only the DL architectures were able to make an accurate forecasting of the system's RUL. Moreover, to the best of the author's knowledge, there is no approach that considers both the fault diagnosis and prognosis tasks, either with a ML or a DL model. The suitability for an on-board implementation is the second requirement. All ML approaches consisted of a low amount of parameters. Moreover, the computations that they required to make a prediction given a new input was very low. These two characteristics make ML models well suited for an on-board fault diagnosis. On the other hand, the very high performance that DL architectures demonstrated in the solution of both FD and FP tasks came with the cost of models with hundred of thousands of parameters and a higher computational complexity. Therefore, the deployment of such DL models would not be feasible in most embedded systems for electric drives, which make current DL approaches unsuitable for an on-board condition monitoring.

In general, ML models for fault diagnosis showed a low flexibility because they required the extraction of frequency and/or time-frequency features like wavelets, which demanded a sound knowledge about the topology of the monitored machine. On the other hand, DL architectures perform an automated feature extraction, which does not require specialized knowledge about the physics of the system, and therefore the adaptation of such a method for a new device would require less effort. The DL approaches indicated with a medium flexibility were tested with only one system. Thus, their flexibility was not completely proven. Only in [104], the flexibility of the proposed approach was validated through two databases. Transfer learning approaches also showed a high flexibility, because an existing network could be adapted to solve a new task without requiring a great amount of data. Moreover, only in [40] a method to label data for the failure prognosis task was proposed. Finally, the introduced approaches for fault diagnosis required CM dedicated hardware such as accelerometers and data acquisition systems, whose implementation in mass-produced mobile systems is not feasible.

The last column of Table 3.1 presents the ranking of each approach. This was obtained by adding the number of requirements that they fulfill. Machine learning models are closer to the objective ranking, since they are giving accurate predictions with compact models. Nevertheless, the approach that has the highest score is a deep learning architecture that makes use of depthwise separable convolutions [104]. Thereby, the computational complexity is reduced while a high performance is kept. Moreover, the flexibility of this model was proven with two datasets. On the other hand, this approach does not cover the FD, nor a method to label data for the FP task is given. Moreover, the resulting model size might not be compact enough to deploy it in an embedded system with low computational resources available. A quantitative comparison between the approaches from the literature and the approach for failure prognosis proposed in this thesis is given in Chapter 6

The following chapter introduces the comprehensive methodology for on-board CM. This approach includes a machine learning model to solve the fault diagnosis task, since ML models have already demonstrated a high accuracy with a low computational complexity for this specific task. On the other hand, the methodology explores two solutions for failure prognosis. First, the performance of a traditional ML method for failure forcasting is improved. Second, generate a DL architecture that is more compact than other existing approaches, while maintaining a high accuracy of the fault prognosis task. Besides the intelligent algorithms for CM, the methodology covers the strategy to collect the data, the signal preprocessing and dimensionality reduction, the data labeling, and the optimization of the algorithms to enable their implementation in embedded systems for electric drives.

Table 3.1: Criteria fulfilled by existing cutting-edge CM algorithms.

		Model performance		Embedded implementation		Flexibility		Application	
	Approach	**Accuracy**	**Task**	**Model size**	**Computational Complexity**	**Flexibility**	**Data labeling for FP**	**CM dedicated Hardware**	**Ranking**
ML	SVM[78]	High	Diagnosis	Compact	Very low	Low	No	Yes	3
	TSVM[90]	High	Diagnosis	Compact	Very low	Low	No	Yes	3
	MLP[70]	High	Diagnosis	Compact	Low	Low	No	Yes	3
	k-NN [105]	High	Diagnosis	Compact	Very low	Low	No	Yes	3
	SVR[42]	Medium	Prognosis	Compact	Very low	Low	No	No	3
	MLP[7]	Medium	Prognosis	Compact	Low	Low	No	No	3
	Kalman filter[40]	Medium	Prognosis	Compact	Very low	Low	Yes	Yes	3
DL	CNN [114]	High	Diagnosis	Big	High	Medium	No	Yes	1
	LSTM[112]	very high	Diagnosis	Very big	Very high	Medium	No	Yes	1
	CNN-LSTM[66]	Very high	Diagnosis	Very big	Very high	Medium	No	Yes	1
	CNN-TL[89]	Very high	Diagnosis	Very big	Very high	High	No	Yes	2
	Bi-LSTM[106]	Very high	Prognosis	Very big	Very high	Medium	No	No	1
	Deep LSTM[116]	Very high	Prognosis	Very big	Very high	Medium	No	No	2
	RBM-LSTM[4]	Very high	Prognosis	Very big	Very high	Medium	No	No	2
	CNN[120]	Very high	Prognosis	Big	High	Medium	No	No	2
	DSCN[104]	Very high	Prognosis	Medium	Low	High	No	No	4
Requirements		High	Both	Compact	Low	High	Yes	No	7

Chapter 4

Comprehensive Methodology for On-board Condition Monitoring

The proposed methodology for on-board monitoring of electric drives is depicted in Figure 4.1. The on-board implementation enables the monitoring of the system condition directly in a computer unit of the system. The main element of this method is the forecasting of the remaining useful life (RUL) of electric drives. This approach focuses on predicting failures of mechanical elements caused mainly by fatigue. Such failures have a progressive occurrence, which enables its forecasting. Two different architectures are proposed for the RUL estimation, the first one is a traditional machine learning model known as Gaussian Process for Regression. The second architecture is a novel deep learning model introduced in this work, namely the Multipath Temporal Convolutional Network. The use case for both models as well as the advantages and disadvantages, are described at the end of this chapter. The fault diagnosis is an additional feature to the method and it is not necessary to estimate the RUL, since the regresion models should be robust enough to forecast the failure independently of the damaged element. A data labeling step is required in case that the training data is unlabeled, which means that the end of useful life is unknown. Within this chapter, every element of the proposed methodology is explained in detail.

4.1 Preparation for the implementation

In the preparation phase, the strategy for data collection is defined. This strategy comprises the definition of the development stage where the data is generated, as well as the possible sources for the dataset construction. This is only a guideline and should be followed if there are no databases available for a specific task. Moreover, at this step, the structure of the dataset is defined. If a database is already available, then only a data cleansing procedure is neccesary to adapt the dataset to the required structure.

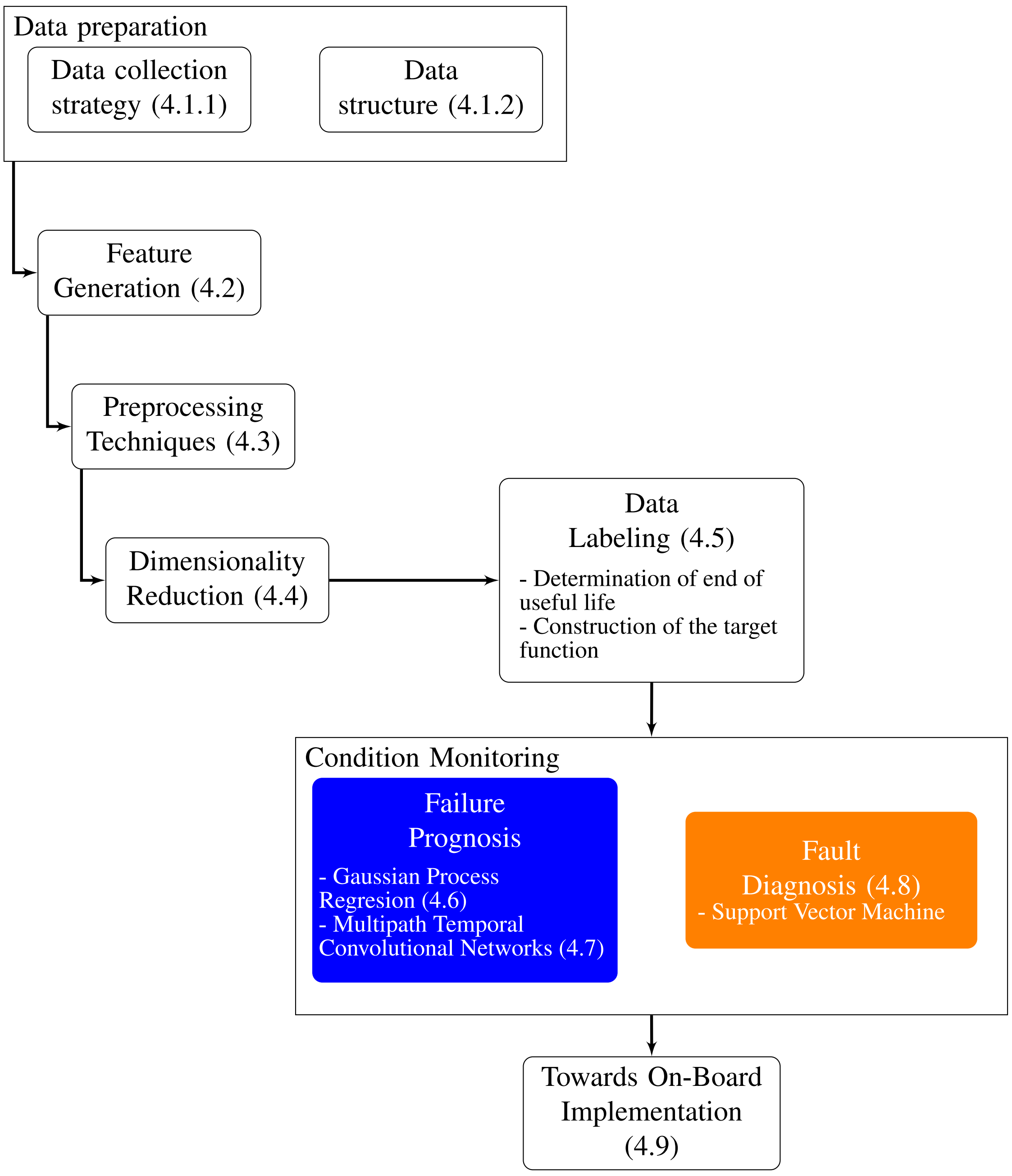

Data preparation
Data collection strategy (4.1.1)
Data structure (4.1.2)
Feature Generation (4.2)
Preprocessing Techniques (4.3)
Dimensionality Reduction (4.4)
Data Labeling (4.5)
- Determination of end of useful life
- Construction of the target function
Condition Monitoring
Failure Prognosis
- Gaussian Process Regresion (4.6)
- Multipath Temporal Convolutional Networks (4.7)
Fault Diagnosis (4.8)
- Support Vector Machine
Towards On-Board Implementation (4.9)

Figure 4.1: Components of the methodology for fault diagnosis and failure prognosis of electric drives.

4.1.1 Data collection strategy

The proposed data collection strategy for each of the main product life cycle stages is the following:

- *Conceptualization*: the analytical models can be used to simulate faults that have a high risk, which means that the combination of the fault severity and frequency of ocurrence is high.
- *Concept validation*: the next step in the design phase is the concept validation through the construction of prototypes. The prototypes go through diverse validation tests such as endurance tests. An advantage given by collecting the data in this stage is that the tested samples suffer many changes before the final design is released for production. Therefore, models trained with this data are more robust to small changes that the electric drives might suffer during the manufacturing processes. The collected data should come directly from the internal sensors of the electric drive, since this the only information that is available during the normal operation of the drives. After the concept has been validated, the production start-up takes place. During the ramp-up phase, validation tests take place to assure the quality of the production process. The data gathered in such tests can be used to retrain the learning models.
- *Market introduction*: the data collected from the systems operated by the end users can serve to refine the algorithms and improve their performance and generalization capability.

Since the creation of accurate analytical models for data generation is a highly complex task, this work focuses only on the data collection within the last two development stages.

4.1.2 Data structure

In a real case scenario, the data does not have the structure required for the development of condition monitoring models. Some common issues with databases are the presence of non numeric values, constant signals, missing values or duplicated variables. Moreover, if the data is gathered from a simulation or a test bench, it is possible that more signals are present that those available during the real operation of the electric drive or that some signals are not present in all tests. Considering these issues, it is required to perform a data cleansing procedure such that the variables with the above-mentioned issues are removed. The resulting database should follow the structure of Table 4.1. The description of each element of the required data structure is given hereunder.

- *ID*: a string or numeric value that is used to identify the system, to which the signal values belong.

Table 4.1: Data structure required for the methodology implementation.

ID	**Running Time**	**Fault Type**	**Sensor Signals**
String / Numeric	Numeric	Categorical / String	Numeric

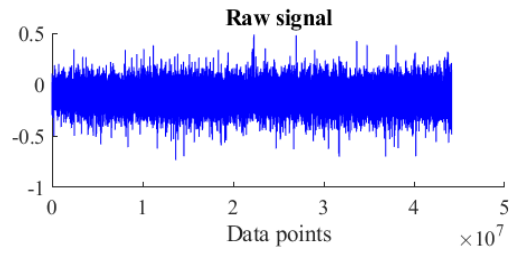
- *Running Time*: a numeric value that indicates the hours or cycles that the the system has been running prior to the execution of the current measurement.
- *Fault Type*: this is an optional value that should be fulfilled if it is neccesary to perform the fault isolation stage of the methodology. It is clear that during the test, the fault type will not be known and only once the test has concluded the drive can be examined to determine the faulty element. Afterwards, this information can be filled into the data array.
- *Sensor Signals*: this is an array with dimension of $\{M, N\}$, where M is the number of samples captured within the measurement and N is the number of sensors monitoring the system. Only those sensor signals that are available during the on-board monitoring and present within all tests should be stored, so that the learning model can extract information that enables the failure forecasting under the normal operation of the electric drive.

Each sample collected according to the data structure presented in Table 4.1 represents a measurement performed at the cycle j. The duration of the measurement and the time span between two measurements can be determined in two ways. In the first option, a measurement with the maximum possible sample frequency takes place each i_t minutes. Around 5 and 15 minutes are adequate for electric drives. Although the system's condition does not show major changes within such a short time span when the drive has no anomaly, once the degradation of an element begins, the condition worsens with an exponential rate. Thus, there might be a considerable change between the sensor values of two measurements performed within intervals larger than fifteen minutes. A second option is to continously monitor the system within the test with a low sampling frequency. For example, to collect only one sample each second.

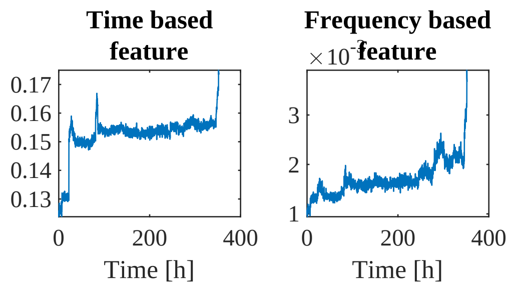
4.2 Feature generation

Once the raw data is correctly structured, it is transformed into time- or frequency-based features. Normally, this step is neccesary when traditional machine learning models are used, due to their inability to perform an automatic feature extraction. Nevertheless, this process is not essential for a deep learning model, such as CNNs, which are able to automatically extract features from raw data. In this case, the feature generation process would enable a smaller model size, since the feature engineering would not be completely performed by the deep learning model. The result is a compact model that can be easily ported into an embedded system. Moreover, the reduction of the number of parameters can also help to avoid problems such as overfitting during the training of the model.

The features described in Section 2.3.1 are generated from the raw signals. This requires to select the length w_l of the array from which the features are extracted. This value is chosen depending on how the data is recorded. If the data is collected with high sampling frequencies each i minutes, the value of w_l should be selected equivalent to the length of the sampled array. On the other hand, if only one sample of each sensor is captured each j seconds, w_l is selected such that the information gathered within several minutes is considered for the feature computation. Figure 4.2a depicts an example of an accelerometer signal that belongs to the IMS bearing dataset [46]. Each bearing in the database is monitored using an accelerometer with a sampling rate of 20kHz. Due to the high sampling frequency, it is possible to extract both time- and frequency-based features. Figure 4.2b depicts two extracted features from the raw signal. It is observed that raw signals provided almost no insight about the change of condition of the bearing with the time. On the other hand, the generated features have a noticeable variation in their amplitudes with the development of the test. Moreover, since the variables are extracted from each measurement, the data array is greatly reduced from almost 44 million samples to only 2156. Nevertheless, due to the increase of the feature space, a dimensionality reduction step is neccesary. This stage is described in detail in Section 4.4.



(a) Raw accelerometer signal.



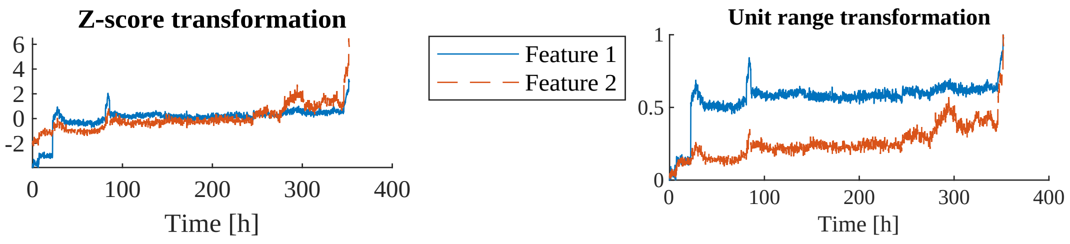
(b) Features extracted from the raw accelerometer signal.

Figure 4.2: Feature generation from a raw signal.

4.3 Signal Preprocessing

Signal preprocessing is an important process prior to the construction and training of any learning model. The preprocessing techniques introduced in this section can be applied to the raw signals or to the extracted features. Through these techniques it is possible to match the ranges of two variables that have very different amplitudes or reduce the noise affecting a signal, among other improvements.

Lets now consider the two extracted features from one of the accelerometers belonging to the IMS bearing dataset. These features are depicted in Figure 4.2b. We can identify that the vaues of the second feature are in a range that is one hundred times smaller than the range of the first variable. If a learning algorithm is trained with these variables without any further processing, then the performance of the model will be influenced only by the second feature and the first variable would be considered as a constant value close to zero. Two techniques to adapt the signals are scaling with the unit range transformation (2.13) and the z-score normalization (2.14). The advantage of the z-score transformation is that not only the amplitude of the signals are adjusted to similar ranges, but also to similar distributions. Figure 4.3a depicts an example of the functionality of the z-score. On the other hand, the unit range conversion only adjusts the values of the signals to the range between zero and one, as seen in Figure 4.3b.



(a) Features standardized with the z-score normalization.

(b) Features scaled with the unit range transformation.

Figure 4.3: Data transformation.

The Z-score normalization considers all data points in the computation of the mean and standard deviation of each signal. Nevertheless, the samples collected within the first cycles of a device and mainly within the cycles close to the failure occurrence deviate greatly from the data points gathered in the healthy state of the electric drive. Moreover, the sensor values within the healthy state might also have considerable deviations among different systems in the database. Thus, the normalization is performed independently for each electric drive and only a reference range of each system is considered for the computation of the Z-score parameters. For this purpose, equation (2.14) is adapted as follows:

$$f_i = \frac{f_i - \overline{X_{ref}}}{\sigma_{X_{ref}}}, \tag{4.1}$$

where $\overline{X_{ref}}$ is the mean of the reference range of each electric drive and $\sigma_{X_{ref}}$ is the standard deviation of the reference range. The reference range comprises a set of data points at the beginning of the lifecycle of the system. This can be as low as 5% of the total life of the electric drive. The most important fact is where the reference range should start, due to adjustment that some elements might suffer during the first cycles of operation.

For instance, sometimes the grease of the gearbox is not correctly distributed among all the mechanical elements. Thus, the mechanism might have an inusual behavior during the first cycles, which could resemble a faulty condition. The *healthy* state of the drive would start when all the components are correctly lubricated. Therefore, it is recommended to start the reference range after around 5 to 10% of the expected system life time. This normalization approach can also be used to normalize the signals for a classification task. For this purpose, the reference range r comprises the data collected from the electric drives without anomalies.

Moreover, the sensors not only capture the values that are related to a physical change of the monitored system but also external disturbances that change the original form of the signal. In order to reduce the effect of these disturbances on the signals, a noise reduction step is performed with a one-side moving average filter (2.15). This filter requires only the selection of a window, which refers to the number of samples that are used to compute the mean. If the window is too large, relevant information of the signal could be lost. Moreover, having a signal with a certain amount of noise, improves the robustness of the models. Therefore, it is suggested to use this smoothing filter with a narrow window.

4.4 Dimensionality Reduction

After the feature generation stage, the dimensionality is increased, which could cause complications such as overfitting. Furthermore, it is intended to implement the algorithms in an embedded system with restricted memory and computing power. Therefore, only the relevant features should be computed online and stored in the memory unit. However, should there be enough computation resources available and a DL architecture is selected, then the dimensionality reduction can be skipped. Two techniques are explored for the dimensionality reduction. The first one is used in classification tasks, where only the identification of the fault type is required. This is the support vector machine recursive feature elimination (SVM-RFE), which is accelerated by using the optimization method of simulated annealing. The second reduction approach focuses on the selection of prognostic parameters that are useful to perform a forecast of the system's time to failure (TTF). The selected features can also be used to train the classifiers for the fault diagnosis. Moreover, this feature selection method is also useful if the data is unlabeled. This approach requires that the database contains tests where the drive's condition degrades gradually.

4.4.1 SVM-RFE with Simulated Annealing Optimization

The SVM-RFE is a wrapper feature selection method that uses the weights computed within the training of an SVM classifier. This approach is described in Section 2.4.1.

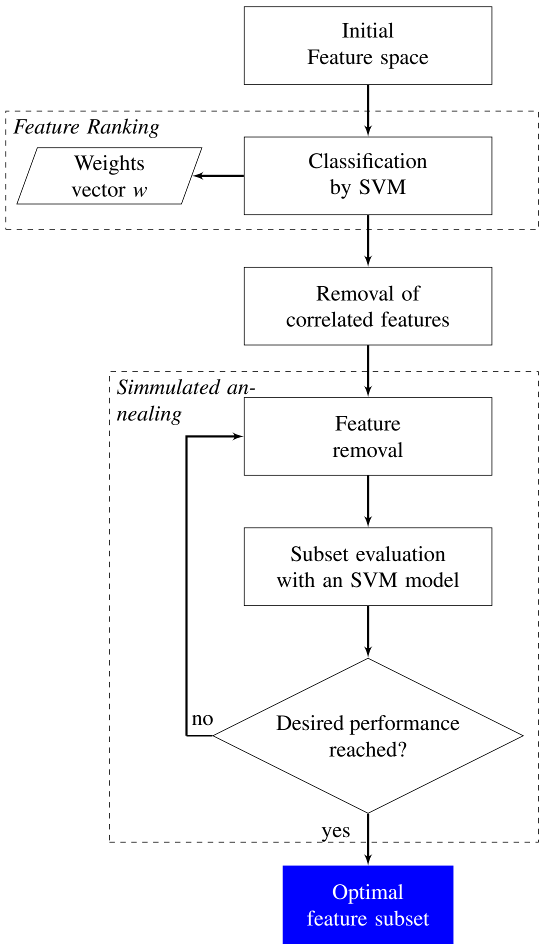


Figure 4.4: SVM-RFE optimized with SA.

Wrapper selection approaches have the advantage that the selected feature subspace maintains the performance of the learning model with less variables or, in most cases, increases the accuracy of the algorithm. Although the ranking of the features spares plenty of search time, high dimensional spaces can still require an extensive search to find the optimal variables. Therefore, the use of simulated annealing (SA) is proposed to accelerate the search for high dimensional problems. The SA approach is introduced in Section 2.8.1. Moreover, we introduced the SVM-RFE with SA optimization method in [101]. In this approach, the energy (E) is equivalent to the classifier's cross validation loss, which is the target metric to be minimized.. Moreover, the feature space size can be considered for the computation of the energy, so that for similar accuracy values, the sub-set with less features is preferred. The energy is then given by:

$$E_i = loss_i + \beta \; n_i / n_S \tag{4.2}$$

where n_S is the initial number of variables, n_i is the feature subset size of the step i and β is a value that determines the importance of the size of the feature set in the energy calculation. For example, if it is desired that with the maximum number of features the accuracy suffers a degradation of 20%, then $\beta = 0.2$. The more important it is to have a final feature subset with less variables, the higher β should be. Besides, it is highly probable that some of the variables are highly correlated, because they are extracted from the same raw signals. The correlation among the variables is given by equation (2.9). In this approach, values with a correlation coefficient larger of ± 0.9 indicate a very strong correlation. It is neccesary to remove the correlated variables once the features have been ranked, in order to have the knowledge about the relevance of each feature, and the less relevant parameters can be removed.

The main advantage of optimizing the SVM-RFE approach with SA is that the search for the optimized parameters can be decreased. This enables time savings when trying to select the variables of a system with a very high dimensionality or when the selection process needs to be repeated several times. For example, when the optimal feature subsets for several classes are searched. Another advantage is that the feature space size can also be considered in the energy calculation, in order to search for a solution with the less features even at the expense of some performance degradation. Figure 4.4 depicts the SVM-RFE approach optimized by SA for feature selection in classification tasks.

To demonstrate the procedure's suitability to reduce high dimensional spaces, we use the *Parkison's Disease (PD) Classification Data Set* [80]. This dataset consists of 756 instances and 754 attributes. The finalization condition is set to be 50 iterations. During the deployment, 37 transitions are accepted. The last accepted transition is also the optimal solution. In Figure 4.5, the score, loss and number of features of each one of the 37 accepted transitions are shown. A seen in this chart, the accuracy is increased by simultaneously decreasing the feature space size to 36. Obtaining an optimized feature subset with a normal wrapper method would have required $3.79e227$ iterations. The SVM-RFE method greatly reduces the number of models that have to be evaluated to only 756. Moreover, the proposed approach requires the evaluation of only 6.6% of the models needed by the SVM-RFE approach, to find a feature subset, which satisfies our requirements of performance and size.

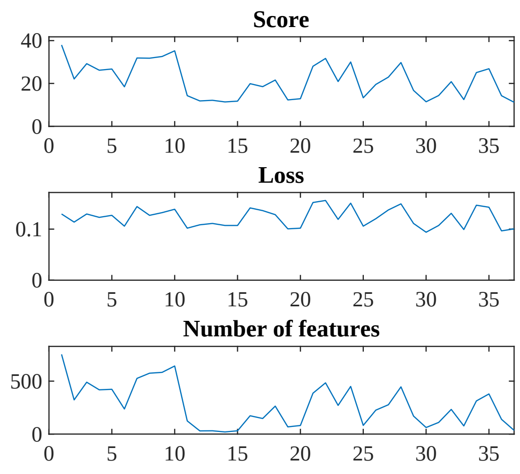


Figure 4.5: Example of SVM-RFE algorithm with SA optimization.

4.4.2 Selection of prognostic parameters

To search an optimized feature subset for the failure forecasting task, the variables are ranked by computing the fitness function 2.19, which depends on the three metrics that quantify the suitability of a feature as a prognosis parameter. These metrics are described in Section 2.4.2. The weighting parameters w_m, w_t and w_p of the fitness function 2.19 are selected such that the metrics are scaled in the range $[0,1]$. An advantage of this procedure is that it is not required that the data is labeled, i.e. that the end of useful life is known. In this instance, the feature selection takes place following the steps described in Algorithm 1.

The factor P_s indicates that the signals with a fitness within Ps percent of the cumulative sum, are selected. A value of $P_s = 0.75$ is proposed as an initial value if the data is not labeled. Otherwise, the value of P_s can be obtained empirically.

Algorithm 1: Feature selection for a regression task.

```
Input: $F_s$: Initial feature space
Input: $N$: Number of features of original feature space
Output: $F_{opt}$: Optimal feature subset
1  Initialization
2  for $i \leftarrow 0$ to $N$ do
3      $m_i \leftarrow monotonicity(F_i)$ (2.16)
4      $t_i \leftarrow trendability(F_i)$ (2.17)
5      $p_i \leftarrow prognosability(F_i)$ (2.18)
6  end
7  $fitness \leftarrow 1/3\,(w_m\, m + w_t\, t + w_p\, p)$
8  $c \leftarrow {}^{fitness}/_{\sum fitness}$
9  $c_s \leftarrow sort(c)$
10 $c_S \leftarrow cumSum(c_s)$
11 $ind \leftarrow getIndex(c_S < P_s)$
12 $F_{opt} \leftarrow F_s[ind]$
```

4.5 Data labeling

Some databases consist only of time to failure experiments, where the system's end of life can be directly inferred as the last measurement before the experiment conclusion. However, there are other databases, which not only have run to failure experiments, but also other tests, where the condition of the system degrades but does not reach the failure occurrence, or where the systems are still healthy. In these situations, the last measurement prior to the conclusion of the experiment cannot be considered as the system's end of life. The data labeling stage is required in this instance. This stage consists of the construction of a health index and the definition of a threshold that indicates the end of useful life of the system. The useful life can be described as the phase in which the systems operate without any major anomaly that could affect their performance over tolerated limits. After obtaining the end of useful life of a system, it is possible to generate the target function that is needed to train the supervised regression models. The last part of this section introduces a method to deal with data inbalance for prognosis tasks.

4.5.1 Health condition determination with the Mahalanobis distance

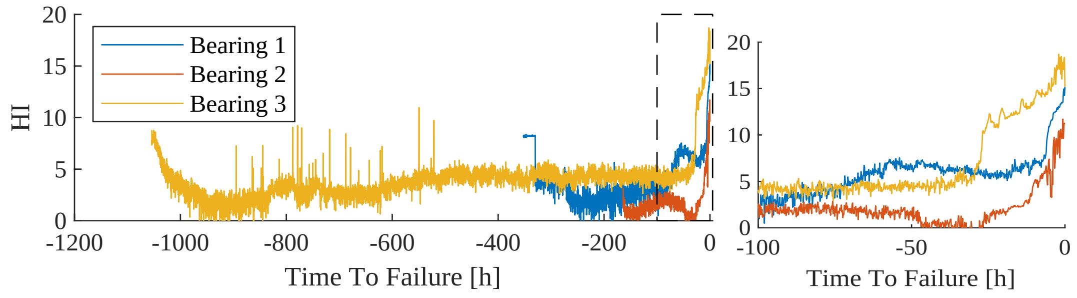
After the feature selection stage, the resulting multi-dimensional feature space F_{opt} is transformed into a one dimensional metric, known as health index (HI), which quantifies the magnitude of a fault. By computing an HI, only one threshold that indicates the end of useful life has to be determined, instead of computing limits for every signal that comprises F_{opt}. The MD (2.10) is selected as the HI within this methodology.

The reference distribution selected for the Mahalanobis distance computation needs to be *robust*, which means that it should be less sensitive to the presence of outliers [11]. The robustness can be assumed if the database is big enough. Within this approach, it is inferred that several degradation curves are available and thus, the robustness is given. Only a subset of the original dataset is considered to be part of the reference distribution. The selected subset consists of the data collected during k_{ref} cycles after the system has elapsed k_{init} cycles. This first k_{init} steps are not included in the reference distribution because, at this time, premature failures can occur. The value of k_{init} is selected as $5-10\%$ of the expected lifetime of the system. From Section 2.2 we know that after the *infant mortality* phase has been overcome, the mechanical systems operate without any major anomalies during long periods. This can be considered as the *healthy* state. Afterwards, an exponential degradation of the mechanical elements take place. It can be assumed that all the systems operate with a healthy state at least until some hours or cycles prior to the failure of the device that achieved the shortest duration in the validation test. Therefore, the value of k_{ref} is given by $k_{ref} = min(FailureTime)/2$. Consequently, the reference distribution T_{ref} is defined as $T_{ref} = F_{opt}(k_{init} : k_{ref})$, and the Mahalanobis distance (MD) is computed with (2.10).

Since the condition degrades with an exponential rate when an element is damaged, the Mahalanobis distance raises exponentially as well. Therefore, the health index is given by a log transformation of the MD, as follows:

$$HI = log(MD). \tag{4.3}$$

To ilustrate the suitability of the Mahalanobis distance to monitor the condition of a system, we use the IMS bearing dataset [46]. The features, which were introduced in Section 2.3.1, were extracted from the raw vibration signals. The original feature subset size was 31. After calculating the fitness for prognosis (2.19) of each variable, only the five most relevant variables were selected, because they constituted 77% of the cummulative sum. The chosen features were used for the Mahalanobis distance computation, in which the base reference was the data collected within the cycles 30-100 of the three experiments. Figure 4.6 displays the HIs of the three tested bearings. The horizontal axis



(a) Complete HI curves. (b) Range $[-1000]$.

Figure 4.6: HI of exemplary bearings.

represents the time to failure. The last value of each test has a TTF of zero and the first measurements have h hours to reach the end of life. As seen in Fig. 4.6a, the HIs of the bearings have similar amplitudes during the healthy state. They also show similar exponential degradation towards the failure occurrence, as seen in Fig. 4.6b. Through this analysis, it is proven how the Mahalanobis distance can be succesfully used to quantify the magnitude of a fault.

4.5.2 Threshold Definition

A common method to determine a fault limit for the detection of malfunctions during the normal life of a mechanical system is the empirical rule [67], where a threshold T is given by:

$$T = m(x) \pm \xi \sigma(x), \tag{4.4}$$

where ξ determines the amount of standard deviations considered for the computation of T. Usually, the empirical rule is used for Gaussian distributed data. The distribution of the HIs differs from a normal shape. Nevertheless, due to the logarithmic transformation, the HIs have a lognormal distribution, as seen in Figure 4.7. This type of distribution is common in reliability systems with positive values only. The log-normal distribution indicates that the logarithmic of a random variable is normally distributed. Therefore, equation (4.4) is suitable for the definition of the thresholds that indicate the start of degradation lim_{deg} and the end of useful life of a system lim_{EOL}. Both limits are given by:

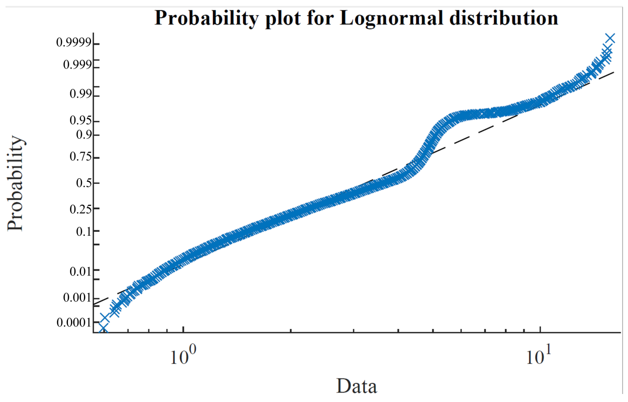


Figure 4.7: Probability plot of tested bearings.

$$\begin{aligned} lim_{deg} &= \mu(x) + \sigma(x), \\ lim_{EOL} &= \mu(x) + 3\sigma(x). \end{aligned} \tag{4.5}$$

For the computation of the thresholds 4.5, the complete dataset is used and not only the reference distribution T_{ref}. Otherwise, the limits would be set too low, and the end of useful life would take place even before the system is slightly damaged.

Figure 4.8 depicts the degradation curve of one of the bearings from the evaluated dataset. The degradation start might have been detected late, because the HI of one of the bearings considered for the computation of the limits has in average higher values

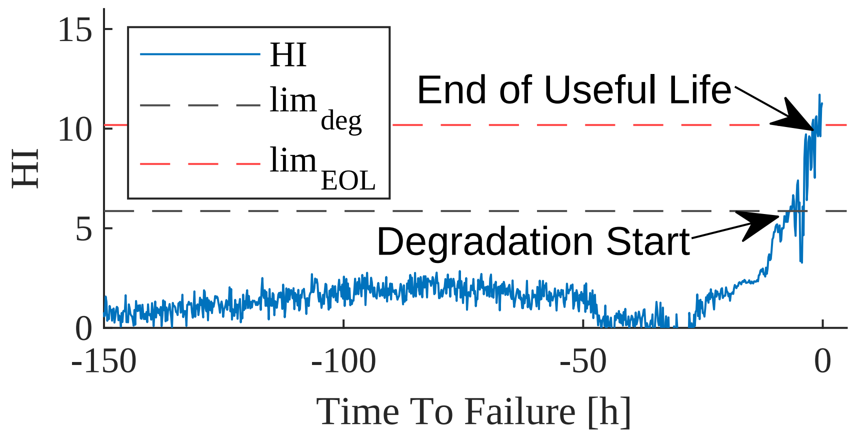


Figure 4.8: HI with selected limits for a tested bearing.

than the other two elements. Therefore, the limits are shifted to a higher amplitude. Nevertheless, the most relevant information is the time at which the end of useful life has been reached. In this example, the end of useful life is reached around 5 hours prior to the failure occurrence.

4.5.3 Target function for RUL estimation

The determination of the end of useful life allows to label the degradation curves of systems that reached the EOL within an experiment. The label corresponding to a set of features computed at the cycle i gives the information of the system's RUL at the $i-th$ cycle. These labels build the target function, which is given by:

$$RUL_{lin} = t_{end} - t \tag{4.6}$$

where t_{end} is the cycle at which the end of useful life is reached and t is the actual running time. By defining the RUL target vector by (4.6), it is assumed that the condition of the machine degrades linearly. However, as seen in Fig. 4.8, the degradation of the mechanical elements is negligible at the start of the test and during long periods. This means that it is not possible to represent the degradation as a linear function. Therefore, a piecewise target function is proposed as the target function, which is given by:

$$RUL_{pwLin} = \begin{cases} maxRUL, & \text{if } RUL_{lin} >= maxRUL \\ RUL_{lin} & \text{if } RUL_{lin} < maxRUL \end{cases}, \tag{4.7}$$

This function has two elements. The first part consists of a constant value equivalent to the maximum RUL that can be forecasted by the regression algorithm. The second part comprises a linear decrease of the RUL until a magnitude of zero is reached, which is equivalent to reaching the end of useful life. It is proposed to compute the maximum number of cycles, which the regression model can predict the EOL in advance by:

$$maxRUL = median\left(DE\right), \tag{4.8}$$

where DE is the time lapse between the degradation start and the end of useful life of all systems contained in the database. The consideration of this period guarantees that when the RUL is lower than $maxRUL$, the deviations in the sensor signals are large enough to enable the forecasting of the TTF. Equation (4.8) is a guide to quickly select a value for $maxRUL$. However, this quantity can be empirically adapted. By selecting a larger magnitude, the regression model should be able to predict a malfunction earlier. Nevertheless, this can be accompanied by an accuracy decrease of the model. On the other hand, the lower the value of $maxRUL$ the better the accuracy of the algorithm, but this results in a belated EOL forecast. Figure 4.9 shows examples of a linear and a piecewise target function.

Another characteristic of the lifecycle of mechanical elements is the exponential degradation, once an element is damaged. This implies that a linear target function is not the most adequate method to represent the degradation of a mechanical system. As demonstrated in our work [61], the performance of the regression model can be boosted with the use of a piecewise exponential target function. Nevertheless, there is a drawback of this approach, which is to obtain the real time that is left before the failure occurrence. In [61], a lookup table is suggested to transform the exponential quantities to a linear range. However, slight changes of an exponential magnitude generate great deviations in the linear signal. Therefore, the linear output of the regression model would not give a rational insight of the remaining useful life. For this reason, the present methodology considers only the use of a piecewise linear target function.

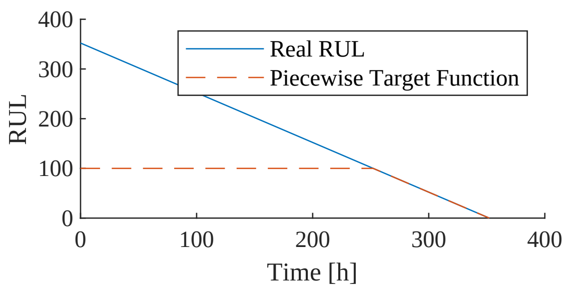


Figure 4.9: Piecewise target function.

4.5.4 Dealing with Data Imbalance

As discussed in the previous sections, the time lapse between any perceptible degration and the failure occurrence is in occasions really short. This, added to long lifecycles of common electric drives result in a data imbalance, which can cause algorithms trained with gradient descent methods to get stuck in a local minima, with a constant output close to $maxRUL$. In case of a non parametric approach such as the GPR, which depends only on the previous observations, the Gaussian Process size and cycle time would increase considerably and the model losses could increase as well, limiting the online implementation of the algorithm. Some methods to cope with the problem of data imbalance are alternative performance metrics such as precision, recall or F1 Score, or sampling strategies such as data over and under-sampling [58]. Within this methodology, a data under-sampling strategy is selected to reduce the negative effect of the data imbalance. The proposed approach is presented in Fig. 4.10.

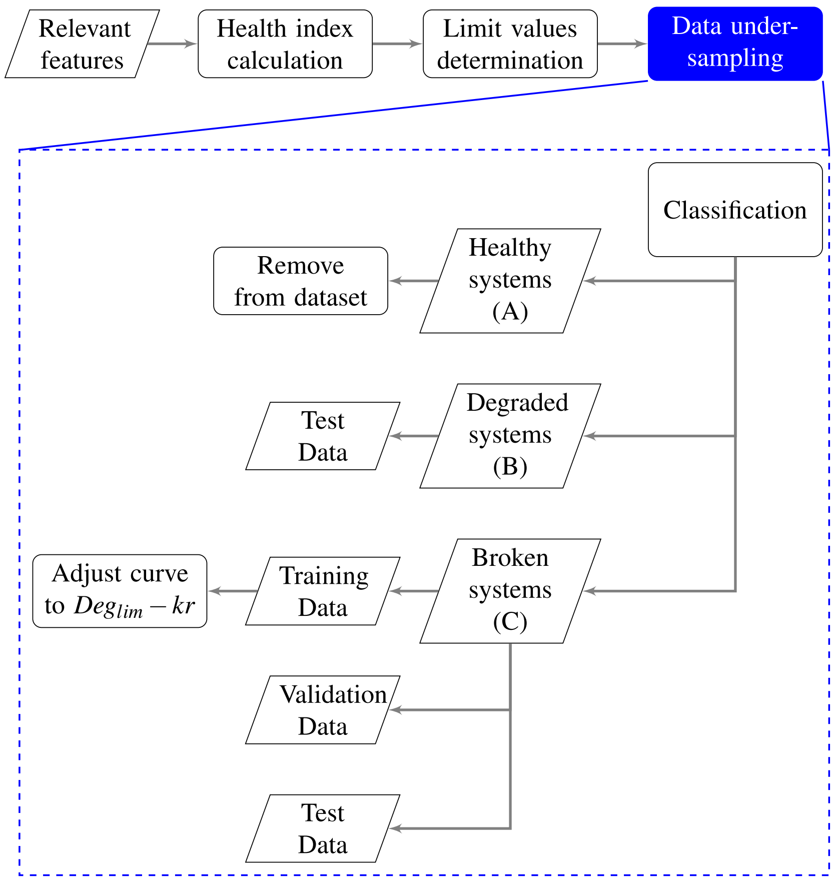


Figure 4.10: Strategy to deal with data imbalance. © 2019 IEEE [61].

This strategy does not only cope with the problematic of data imbalance, but also gives a guideline on how to proceed with systems, which do not break down within the test. To this end, the drives are classified in three groups: A, B, and C.

- *Class A* are systems, which ended the validation test without any anomaly. The drives belonging to class *A* are removed from the dataset and are just classified as healthy systems.

- *Class B* corresponds to systems with a final health index between the degradation and failure limits. The samples classified as category *B* are used to further test the robustness of the algorithm, once it has been trained and validated.

- *Class C* are those drive units, which reached the end of life threshold within the test. The drives belonging to the category *C* are selected to train, test and validate the regression algorithm.

A second undersampling method for the training dataset is applied to reduce the amount of information generated when a system is operating normally. Here, only the measurement's segment between the degradation start and the failure occurrence are considered. However, this would not give the regression algorithm information about the machine's behavior with a healthy condition. Thus, a segment corresponding to the measurement prior to the degradation start is integrated to the training dataset as well. This segment has a length of kr cycles, which can be selected equivalent to $maxRUL$. An example of an undersampled curve is shown in Figure 4.11.

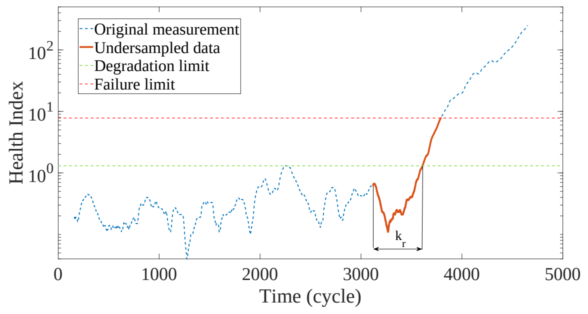


Figure 4.11: Undersample Strategy for C-class drives. © 2019 IEEE [61].

4.6 Gaussian Process Regression for failure prediction

Once the data has been preprocessed and labeled, the task of the RUL estimation is solved by a supervised regression model, such as the Gaussian Process Regression (GPR) model. The GPR algorithm is widely used in time series prediction, where it estimates future outcomes using the knowledge gained from the present and previous inputs. GPR models are probabilistic algorithms, whose output is the expected value for the input vector x based on the knowledge gathered in previous experiments. An advantage of these type of models is that not only the prediction is given, but also a confidence interval. This is useful to know the certainty of the output of the algorithm. A narrow interval is a positive sign, since it indicates a high probability of a correct output. Even if the prediction is wrong, the predicted value will probably have a small deviation from the target value.

The proposed process to train and test a GPR model for RUL prediction is depicted in Figure 4.12. The first step is the preparation of the training dataset, which comprises the input vectors x_{train} and their corresponding labels y_{train}. Here, it should be decided whether the selected relevant features or the HIs are used as input to train the GPR. On the one hand, the HIs have already identified the degradation of a single system with regard to a basis distribution. On the other hand, the features retain more information about the system idependently of the other drives in the database. Moreover, the features are noisier, which improves the robustness of the model and its capability for generalization. The input vector x_{train} is given by $x_{train} = x_1, x_2, ..., x_n$, where n is the number of data points in the training dataset and each vector x_i has a size $\{p, m\}$, where m is the number of features and p is equivalent to one, since the GPR performs a prediction for a single time step. The train labels y_{train} are given by $y_{train} = y_1, y_2, ..., y_n$, where each label y_i is a scalar.

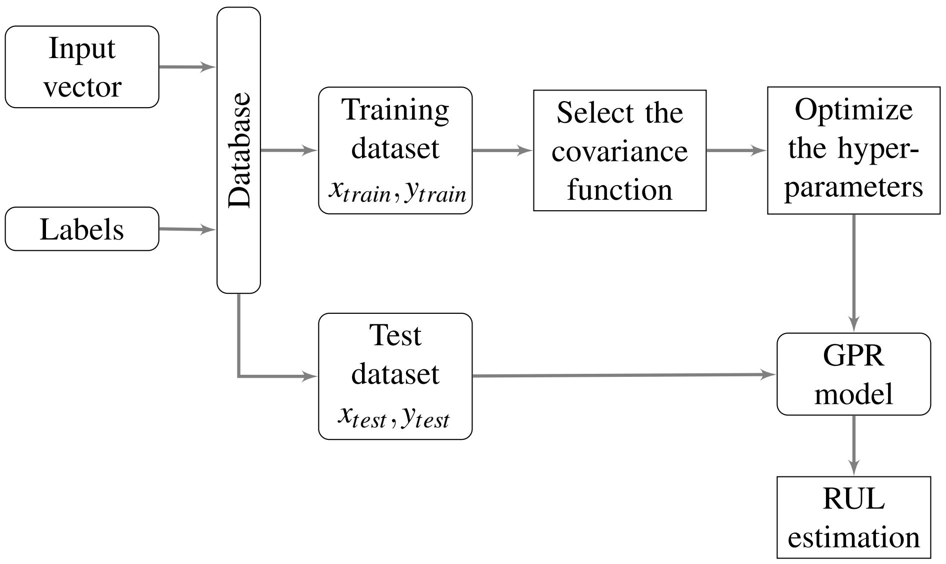

Input vector
Labels
Database
Training dataset x_{train}, y_{train}
Select the covariance function
Optimize the hyper-parameters
Test dataset x_{test}, y_{test}
GPR model
RUL estimation

Figure 4.12: Process to train a GPR model for RUL estimation

Prior to the training of the GPR model, a covariance function $k(x_n, x_m)$ has to be selected. There are several standard kernel functions that can be selected, some of which are depicted in Table 2.1. Alternatively, a new covariance function can be built. Due to the exponential behavior of the degradation curves, the exponential kernel can be a suitable choice as a covariance function. This function has only two hyperparameters: the length scale σ_l and the signal standard deviation σ_f. These hyperparameters are denoted by θ and are initialized by (2.28).

The training of the GPR model consists on finding the values of the kernel hyperparameters θ, and the noise variance σ_n^2. The hyperparameter optimization takes place by maximizing the log marginal likelihood (2.38). Even with the data undersampling strategy explained in Section 4.5.4, the remaining training dataset might still be too large and an exact maximization of (2.38) might not be possible due to the high computation complexity. In that case, an approximation method of Table 2.2 can be implemented for the hyperparameter optimization. After the hyperparameters θ have been optimized, the RUL prediction for a new set of data points takes place by (2.37). To test the performance of the GPR model the test dataset is used. The inputs x_{test} and the labels y_{test} have the same dimensions than x_{train} and y_{train} respectively.

The process for a new prediction is the following:

1. A new input vector x_{test_i} is given to the GPR.
2. In the model, x_{test_i} is compared with all the train vectors x_{train} and their respective label y_{train} as indicated in (2.37).
3. The GPR outcome is the mean RUL prediction for the given input x_i compared with the train data. The confidence interval can be interpreted as the minimum and maximum RUL values for vectors in x_{train} that are similar to x_{test_i}.

Figure 4.13 depicts an example of how a trained GPR model predicts the RUL of a electric drive at each time step i. This set of predictions create the estimated function y_{pred}. The outcome of the GPR can also be visualized as a set of distribution curves that together build the function y_{pred} with a confidence interval.

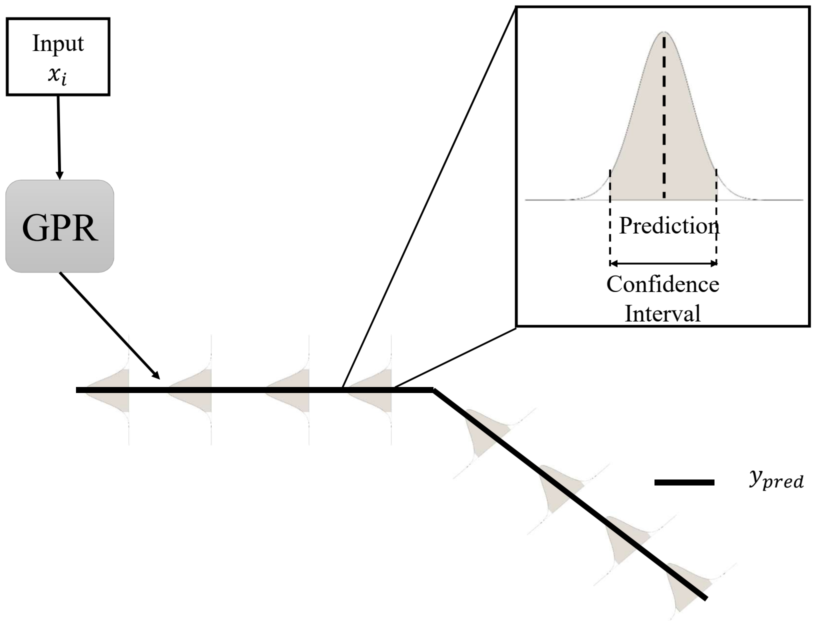


Figure 4.13: Illustration of a GPR model for RUL estimation.

4.7 Framework for failure forecasting with Multipath Temporal Convolutional Networks

The GPR is an effective method to forecast the RUL of mechanical systems and it is very useful for embedded systems with low computing power and memory available. However, many preprocessing steps have to take place in order to achieve a good estimation. An alternative to traditional machine learning architectures are deep learning models, which have shown a great performance for sequence modeling tasks, such as the RUL estimation. Deep learning models are a very powerful tool for automatic feature engineering. This characteristic not only reduces the work during the data preparation but also enhances the flexibility of the methodology. Moreover, deep learning models have a better performance than traditional machine learning architectures. One of the reasons behind this afirmation is that DL models are able to find features in the raw signals, which cannot be extracted during the manual feature engineering. Even though the DL architectures might seem an obvious selection over traditional machine learning models such as the GPR, their implementation in an embedded system is limited due to the size of the models and the great amount of operations that take place to perform a prediction. Considering the advantages and limitations of deep learning models, a DL framework for RUL estimation that is suitable for an on-board implementation is introduced in this section. The proposed framework is depicted in Figure 4.14. The core element of the framework is a novel deep learning architecture named Multipath Temporal Convolutional Network (MTCN), which we presented in [61]. The MTCN is based on the residual blocks of the TCN model developed by authors in [6]. The use of this framework is dedicated to applications, where the are enough computing resources for additional functions. A discussion of when to use which type of model for the RUL estimation is provided in Section 4.9.3.

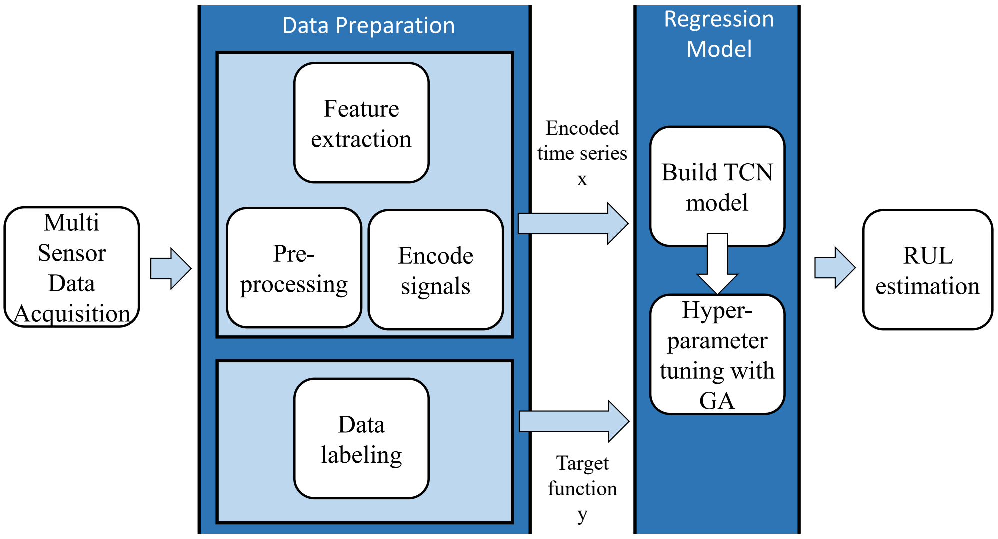


Figure 4.14: Framework for RUL estimation based on Multipath Temporal Convolutional Networks.

This framework has two main stages: the data preparation, and the generation and training of the regression model. The first part has three steps which have been explained in detail throughout this chapter. In the following subsections, the importance of each stage within this framework is described.

4.7.1 Data Preparation

Feature Extraction for deep learning

This is a process that can be avoided with the use of a DL architecture. Nonetheless, a feature extraction before the construction of the deep architecture enables the reduction of the network size and accelerates the training time. Since the objective is that the model is ported into an embedded system, this step is relevant mainly when dealing with raw signals with thousands of data points. For example, deep learning architectures might require hundreds of thousands of parameters to extract relevant features from signals that were sampled with high frequencies, such as accelerometers, which are commonly used for condition monitoring. The process of feature generation is introduced in Section 4.2. On the other hand, a manual feature reduction step is not neccesary anymore, because the filters within the architecture are in charge of identifying the relevant attributes from the input feature maps.

Signal preprocessing

The signal preprocessing is a very important step within the data preparation, as already explained in Section 4.3. The preprocessing stage considers the signal normalization procedure detailed in Algorithm 4.1 and the scaling of the standardized signals in the range $[0,1]$ by (2.13). A noise reduction of the signals is not performed within this deep learning framework. By avoiding this step, the MTCN can generalize better despite the presence of disturbances in the sensor signals.

Encode signals

The objective of sequential modeling tasks is to predict the outputs $y_0,...,y_T$ at each step for a given sequence input $x_0,...,x_T$. Therefore, the preprocessed signals or extracted features should be transformed into encoded time series, which have a size of $\{W,C\}$, where C are the channels or features and W is the number of time steps contained in the feature map. One great advantage of working with embedded time series instead of vectors with single time steps is that the learning algorithm is less sensitive to ouliers. Only the increase on amplitude in several time steps could be related to an upcoming malfunction. An example of an embedded time series is depicted in Figure 4.15.

Data labeling for DL framework

In case that the end of useful life is already defined for each measurement in the dataset, this information is used to build the target function y_{target}. Otherwise, it is necessary to determine the thresholds that indicate the end of useful life, to then be able to generate the target function. The data labeling procedure was explained in detail in Section 4.5.

4.7.2 Multipath Temporal Convolutional Network for RUL estimation

Common architectures for sequential modeling are the recurrent neural networks and the long-short term memory models. The RNNs normally face problematics such as exploding or vanishing gradients during the training of the model. In LSTMs this problematic is minimized by using memory gates. Nevertheless, the prediction of a new set of inputs requires a sequential amount of operations, which cannot be parallelized. On the other hand, CNNs have demonstrated a great performance in sequential tasks as well. They are suitable for parallelization, since the operations to compute each output channel are independent. A CNN architecture that has a great efficiency in sequential modelling is the TCN model. This has three main features: the use of zero padding to map an input into an output of the same dimension; the usage of causal convolutions to avoid leakage of information from future to past; and the employment of dilated convolutions to have a wider receptive field with less parameters. These characteristics are explained in detail in Section 2.6.2. Figure 4.15 shows how the residual blocks of the TCN are used to construct an architecture that enables the failure prediction of electric drives.

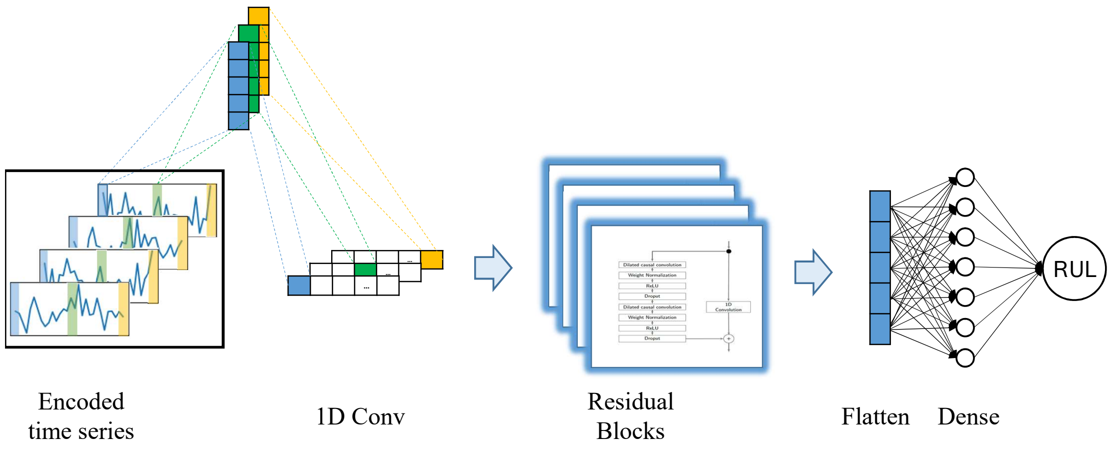


Figure 4.15: TCN architecture for RUL prediction.

MTCN architecture

The architecture of the proposed Multipath Temporal Convolutional Network was published in our work [62] and it is depicted in Figure 4.16. As its name suggests, the MTCN is composed of two paths. The first path has the same basis structure as the TCN for RUL depicted in Figure 4.15. The encoded time series generated in the data preparation stage is the input to the first path. The dimension W is selected as the equivalent to the receptive field RF (2.54) of the architecture. The dimension C is equal to the number of features. The input images are connected to a 1D-convolutional layer, which works as the first feature extraction filter. Since a zero padding is used, the width W of the feature map remains constant. Moreover, the number of channels of the output feature map C_{out} is equal to the number of kernels k. This first 1D convolutional layer is activated by a ReLU function (Table 2.3). The output of the first convolutional layer is connected to a series of residual blocks, whose structure is depicted in Figure 2.10. The number of residual blocks depends on the dilation rate, which is determined in the hyperparameter optimization approach.

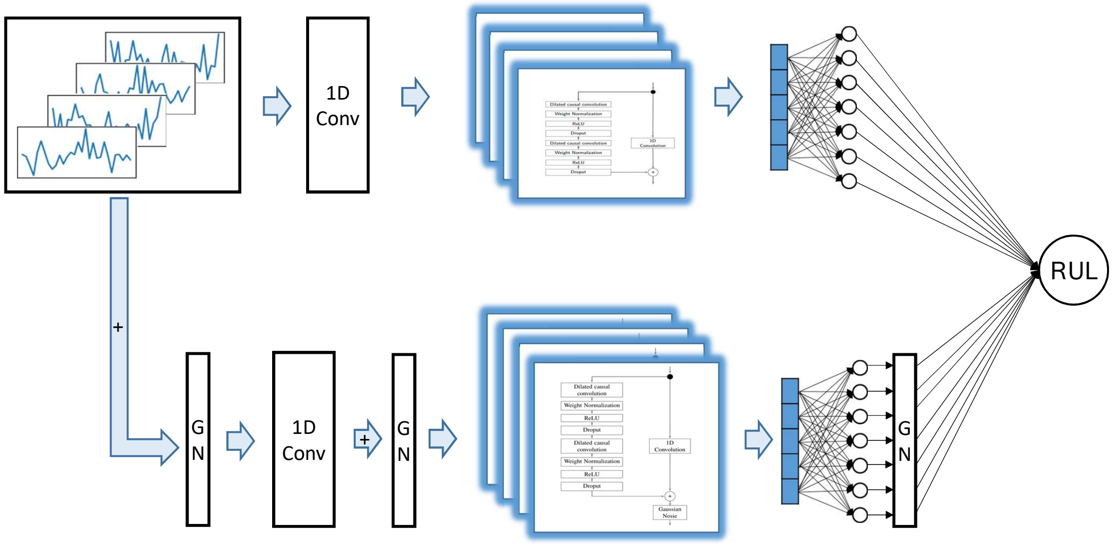


Figure 4.16: Multipath Temporal Convolutional Network. © 2020 IEEE [62].

The output of every residual block has the same dimension as its input. Thus, the output of the last residual block has a size of $\{W, k\}$. Afterwards, a flatten layer is used to reshape the feature map delivered by the last residual block as a vector with dimension $\{n_{flatten} \cdot k, 1\}$, where $n_{flatten}$ indicates the number of elements of the feature map that are flattened. Then, the vector is connected to a dense layer. The number of neurons of the dense layer n_{dense} is another hyperparameter that has to be optimized.

The second path runs parallel to the first one and has the same architecture. The input of this path is the same embedded time series, which is connected to a Gaussian noise layer prior to the first convolutional layer. After the initial 1D convolution, the resulting feature map is also added with Gaussian noise. The output is connected to a series of modified residual blocks. The difference to the blocks proposed in [6] is that the output is connected to a Gaussian noise layer. Figure 4.17 depicts the modified residual block. Similarly to the first path, the output of the last residual block is flattened to connect it to a dense layer. The output of the dense layer is summed with Gaussian noise. Finally, both paths are concatenated and connected to the output dense layer with a linear activation function, which delivers the estimated RUL of the mechanical system. This architecture enhances the generalization capability of the model by reducing its sensibility to noisy feature maps. The added Gaussian noise is centered in zero and has a standard deviation g_N, where $0 < g_N < 0.5$. Thus, there are only a few outliers with an absolute magnitude greater than 1. The Gaussian noise layers are active only during the training of the model. When the trained model is used to predict the RUL with a new feature map, the architecture acts as a two path network without adding noise to the layers.

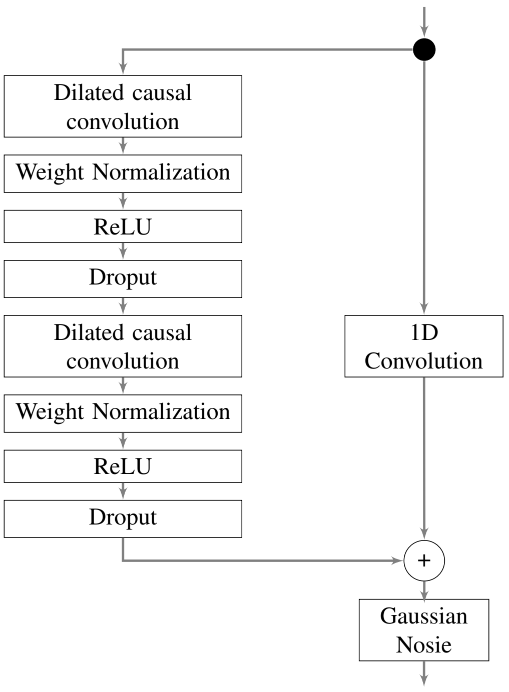


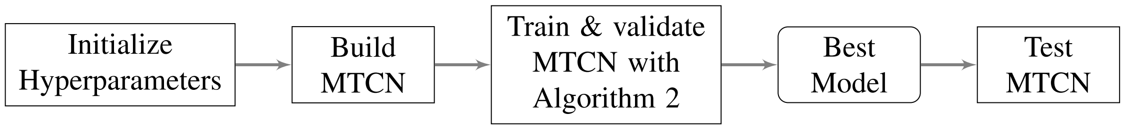
Figure 4.17: Modified residual block.

One can argue that the noise inclusion can be performed directly in the first path and thus, avoid two parallel architectures that merge at the end. Nevertheless, this has the negative effect that the model requires more epochs to converge and frequently, it can not even converge. The two paths of the MTCN perform two independent feature extraction stages, which means that different relevant features are identified at each path. This simplifies the task of the RUL estimation by the dense layer, which enables dense layers with less neurons compared to TCN models. Besides, the experimental studies presented in this work show that the MTCN requires less filters or less residual blocks to achieve a higher accuracy than TCNs.

In conclusion, this novel architecture has two main advantages over the traditional TCN: a robust model that has a better performance with unseen samples; and a more compact architecture, which is relevant for the embedded implementation.

MTCN training

The process to train, validate and test the MTCN model is depicted in Figure 4.18.


Initialize Hyperparameters
Build MTCN
Train & validate MTCN with Algorithm 2
Best Model
Test MTCN
Figure 4.18: Process to train the MTCN model for RUL prediction

Prior to the construction of the MTCN model, the hyperparameters have to be initialized. These parameters determine the architecture of the model and also include coefficients that control the training of the network. We define in total 10 hyperparameters, which are:

- **Kernel size** (ks): is the width of a filter and represents the number of time steps that the kernel covers.
- **Dilation rate** (dil_{RL}): refers to the dilation that the last residual block should have. The dilation of each layer is given by: 2^d, with $d = 0, 1, 2, .., dil_{RL}$. This hyperparameter, together with the kernel size, determines the network's RF (2.54).
- **Number of filters** (k): amount of filters of each layer. For this architecture, the number of filters remains constant for all layers.
- **Number of neurons of dense layer** (n_{dense}): determines the size of the dense layer.
- **Number of elements of the flatten layer** ($n_{flatten}$): the number of *time steps* of the last feature map that should be connected to the dense layer.
- **Dropout rate** (dr): as indicated in Section 2.6.2, the dilated causal convolutions within the residual blocks are regularized by a *dropout* method, which reduces the possibility of overfitting. The dropout temporarily removes an amount of parameters together with their connections [94]. The dropout rate indicates the percentage of parameters that are "dropped" at each training epoch.
- **Learning rate** (η): controls how much the weights are adapted regarding the loss gradient.
- **Batch size** (bs): is the number of feature maps that are given simultaneously as an input to the architecture during the training procedure.
- **Window displacement** (w_t): indicates the shift in cycles between the starting points of two feature maps.
- **Standard deviation of the Gaussian noise distribution** (g_N): the greater g_N the greater the amplitude of the noise that it is injected to the second path.

The hyperparameters can be initialized by defining ranges of each coefficient and then randomly selecting a value within these limits for each parameter. This initialization is performed randomly, since the hyperparameters (HP) will be afterwards optimized with a genetic algorithm. After the selection of the parameters, the MTCN model can be built and trained. The training procedure is described in Algorithm 2. The inputs to this algorithm are the training $\{x_{train}, y_{train}\}$ and validation $\{y_{val}, y_{val}\}$ datasets. The feature maps $\{x_{train}, x_{val}\}$ are 3 dimensional arrays which have a size of $\{K, W, C\}$, where K is the number of feature maps of each dataset and W, C are the time steps and number of features of each image. W and C are constant for both datasets. At the beginning of the

Algorithm 2: Train MTCN

```
Input: $MTCN_{model}$: MTCN model
Input: $x_{train}, y_{train}$: Train dataset
Input: $x_{val}, y_{val}$ Validation dataset
Input: $epochs$: Number of epochs
Input: $bs$: batch size
Input: $\eta$: learning rate
Output: $MTCN_{out}$: Best trained MTCN model
Output: $E_{best}$: validation loss of $MTCN_{model}$
Output: $n_{par}$: number of parameters of $MTCN_{model}$
1  Initialization
2  $E_{best} \leftarrow 10000$
3  $L_{Tr} \leftarrow K_{train}/bs$
4  $Initialize\ the\ filter\ weights\ by\ Xavier\ initialization$
5  for $i \leftarrow 0$ to $epochs$ do
6      for $k \leftarrow 0$ to $L_{Tr} - bs$ by $bs$ do
7          $MTCN_{model} \leftarrow Adapt\ weights(MTCN_{model}, x_{train}[k : k+bs], y_{train}[k : k+bs], \eta)$
8      end
9      $\hat{y}_{val} \leftarrow Predict(MTCN_{model}, x_{val})$
10     $E_{val} \leftarrow RMSE(y_{val}, \hat{y}_{val})$
11     if $E_{val} < E_{best}$ then
12         $MTCN_{out} \leftarrow Model\ checkpoint(MTCN_{model})$
13         $E_{best} \leftarrow E_{val}$
14     end
15 end
16 $n_{par} \leftarrow get\ number\ of\ parameters(MTCN_{out})$
```

algorithm, we need to define the number of epochs, which is the number of iterations that will be performed with the whole training dataset. The number of iterations L_{Tr} for each epoch is given by dividing the number of images of the training dataset by the batch size. After the inputs and initial values have been defined, the parameters of the model are initialized by the Xavier weight initialization [21]. Then, the gradient $\nabla E(\underline{w})$ of the quadratic error is computed with the back propagation algorithm and the filter weights are adjusted on basis of this gradient and the learning rate η. After all the batches have been used for the weight update, the validation loss E_{val} is computed by the root mean squared error loss function, which is given by (4.9). If the new validation loss is lower than the best loss E_{best} until that iteration, a model checkpoint is created and the parameters are stored. This assures that the model with the lowest validation loss is delivered at the end of this algorithm. The procedure is repeated until the defined number of epochs is reached. The performance of the best obtained model is assessed with the test dataset $\{x_{test}, y_{test}\}$ in order to verify the generalization capability of the model with unseen data.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_{target_i} - y_{predicted_i})^2}. \tag{4.9}$$

4.7.3 Automatic hyperparameter tuning with a Genetic Algorithm

The improvement of the model performance can be achieved by modifying the hyperparameters. Lets assume that we only have two possible values for each one of the ten HPs that were defined in the previous section. Then, we would require to evaluate 2^{10} models in order to select the optimal solution. Nevertheless, the defined HP have continuous values, which means that, theoretically, each hyperparameter has infinite possible values. Thus, a manual tuning of the parameters is not feasible. A genetic algorithm is proposed to select an optimized set of hyperparameters, which can improve the accuracy of the MTCN model. The *optimized* model is a local optimum, whose performance should fulfill our requirements of accuracy and size of the network. As explained in Section 2.8.2, the GA has five main steps, which are the creation of the initial population, the evaluation of the fitness, the selection of the parents, the crossover to generate offspring and the mutation. This section details how each stage of the algorithm is performed to optimize the hyperparameters of the MTCN model.

Population creation

First of all, the ranges, from which the HPs can take a possible value, are defined. The selection of the limits is based on knowledge about each parameter or about possible values for certain HPs. For example, the dropout rate can be only in the range $\{0, 1\}$. Based on the defined ranges, the initial population P with n_{ind} individuals is generated as in Algorithm 3. Each individual of the population is composed of a determined amount of genes n_g, which for this application are the hyperparameters.

Evaluation

The second step is the evaluation of the population's fitness. First, an MTCN model is built with the hyperparameters of the i-th individual of the population. The model is trained following Algorithm 2, which outputs the validation loss E_{val} and the number of parameters n_{par} of the model. Afterwards, the fitness of the individual is given by:

$$f = E_{val} + k_p \cdot \; n_{par}, \tag{4.10}$$

where k_p is a weighting constant that indicates the importance of the number of parameters in the fitness computation. Integrating this parameter enables to obtain a compact

```
Algorithm 3: Genetic Algorithm: Create Population
  Input: $n_g$: number of genes
  Input: $n_{ind}$: amount of individuals per population
  Input: $l_{up}$: vector with the upper limit of each gene
  Input: $l_{low}$: vector with the lower limit of each gene
  Output: $P$: population
1 Initialization
2 $P \leftarrow ()$
3 for $i \leftarrow 0$ to $n_{ind}$ do
4     $r \leftarrow Generate\ random\ vector(0,1,n_g)$
5     $individual \leftarrow r \cdot (l_{up} - l_{low}) + l_{low}$
6     $P \leftarrow append(individual)$
7 end
```

model for the posterior embedded implementation. If only the accuracy of the given model is important, then $k_p = 0$. Afterwards, the fitness of the individual is appended to the fitness of the population F. This process is repeated until all the individuals have been evaluated. The fitness of the population is stored in the variable F_{tot}.

Selection

In the third step, a selection of the so-called *parents* S takes place. Here, each individual of the population has a positive probability of being selected. This probability P_k depends on the inverse fitness. Thus, the individuals with lower fitness have greater chances to be chosen. In our approach, the parents are selected following Algorithm 4.

Crossover

The fourth step is the crossover, where the information of the selected *parents* is mixed to create new *offsprings* O. We perform a random distributed pairing, in which each parent can be paired with another parent with the same probability, independently of their fitness. Within this approach the amount of genes and which genes are transmitted from each parent to the offsprings are randomly chosen for each parents pairing, instead of defining a crossover point. This procedure is chosen to avoid that the offsprings always share the same adjacent genes, and thus, to evaluate more diverse solutions. The parents and offsprings constitute the new population $P_{new} = [S, O]$.

Mutation

In order to explore new solutions, some genes of P_{new} are randomly mutated. Algorithm 5 depicts how the mutated population is created. After the deployment of this algorithm,

Algorithm 4: Genetic algorithm: Parents selection

```
Input: $F$: fitness of all elements within the population
Input: $n_{ind}$: amount of individuals in the population
Output: $S$: parents
1  Initialization
2  $S \leftarrow ()$
3  $F_{inv} \leftarrow {}^{1}/_{F}$
4  $F_x \leftarrow {}^{F_{inv}}/_{\sum F_{inv}}$
5  $P_k \leftarrow cumulative\ sum(F_x)$
6  $n_S \leftarrow {}^{n_{ind}}/_{2}$
7  while $i < n_S$ do
8      $r \leftarrow Generate\ random\ value(0,1)$
9      $idx \leftarrow find\ nearest(r, P_k)$
10     if $isin(idx, S) == False$ then
11         $S \leftarrow append(idx)$
12         $i \leftarrow size(S)$
13     end
14 end
```

the process is repeated from the second step, the fitness evaluation. The search of the optimal hyperparameters ends when n iterations have been performed. Finally, the optimized set oh hyperparameters Ind_{opt} is selected by:

$$Ind_{opt} = argmin(F_{tot}). \tag{4.11}$$

The genes of the optimized individual are selected as the optimal hyperparameters for the MTCN model and the procedure concludes. The same steps are followed in case that the HPs of a TCN architecture should be tuned.

Overview of the complete algorithm

Figure 4.19 depicts all the above described steps of the GA to optimize the TCN and MTCN hyperparameters. Moreover, each step is exemplified for a better understanding of the algorithm. In this example, a population of four individuals with 9 hyperparameters is created. Then, four different TCN architectures are built and trained. Their fitness is then evaluated. Afterwards, the parents are selected following the roulette wheel selection procedure. As seen in Fig. 4.19, the models with a better fitness cover a greater area of the roulette and have thus, more probabilities of being selected. The individuals one and four are selected as the parents. Then, their genetic information is combined to create two offsprings. The two parents and two offsprings compose the new population. In the last step, a random amount of genes determined by the mutation rate are adapted.

Algorithm 5: Genetic Algorithm: Mutation

```
   Input: $P_{new}$: new population
   Input: $g_{pop}$: total number of genes of the population
   Input: $mut_{rate}$: mutation rate
   Input: $\Delta mut_{rate}$: mutation rate change percentage
   Output: $P$: mutated population
   Output: $mut_{rate}$: adapted mutation rate
 1 Initialization
 2 $P_a \leftarrow Create\ Population$
 3 $g_{mut} \leftarrow mut_{rate} \cdot g_{pop}$
 4 $g_{ind} \leftarrow Generate\ random\ vector(1, g_{pop}, g_{mut})$
 5 $P_{new}[g_{ind}] \leftarrow P_a[g_{ind}]$
 6 $P \leftarrow P_{new}$
 7 $mut_{rate} = mut_{rate} \cdot \Delta mut_{rate}$
```

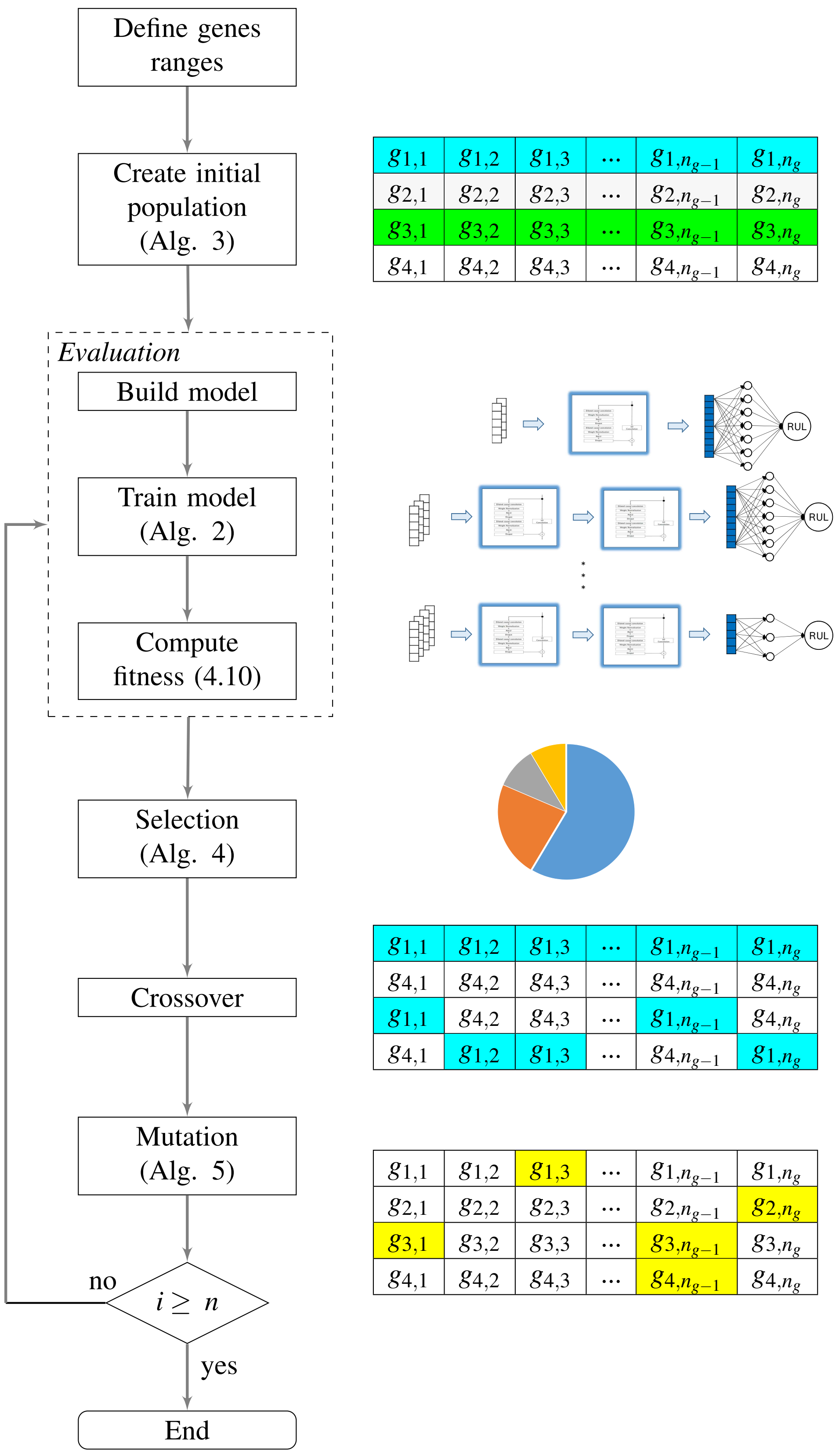
If the termination condition is not fulfilled, in this case the total number of iterations has not been reached yet, the new population is used to create the new MTCN models and the process is repeated until all the required populations have been evaluated.

4.7.4 Regularization technique for generalization improvement

In Section 2.7, data augmentation techniques for time series, such as window cropping and noise injection, have been introduced. Since the inputs to the MTCN architecture are embedded time series, it is not neccesary to crop certain regions but just to randomly duplicate feature maps. Moreover, the feature maps are already added with noise at the second path of the MTCN architecture. Therefore, instead of injecting disturbances to the duplicated images, a determined amount of time steps are substituted by zeros. This data augmentation technique is given by Algorithm 6, and it is applied for both the train and validation sets. By training the MTCN with inputs with zero-valued rows, the model is better suited to predict the RUL of short measurements with zero padding to match the size of the receptive field. This approach not only works as a regularization method but can be also used as data augmentation tehcnique, since it appends new feature maps to the dataset.

4.8 Fault diagnosis with a multiclass SVM

The previous sections have introduced methods to identify, quantify and forecast a malfunction of an electric drive. In occasions, it is required not only to forecast a malfunction but also to locate its root cause while the system is still in operation. Within this section, a fault diagnosis based on a multiclass SVM is presented.



$g_{1,1}$	$g_{1,2}$	$g_{1,3}$	...	g_{1,n_g-1}	g_{1,n_g}
$g_{2,1}$	$g_{2,2}$	$g_{2,3}$	...	g_{2,n_g-1}	g_{2,n_g}
$g_{3,1}$	$g_{3,2}$	$g_{3,3}$	...	g_{3,n_g-1}	g_{3,n_g}
$g_{4,1}$	$g_{4,2}$	$g_{4,3}$	...	g_{4,n_g-1}	g_{4,n_g}

$g_{1,1}$	$g_{1,2}$	$g_{1,3}$	...	g_{1,n_g-1}	g_{1,n_g}
$g_{4,1}$	$g_{4,2}$	$g_{4,3}$	...	g_{4,n_g-1}	g_{4,n_g}
$g_{1,1}$	$g_{4,2}$	$g_{4,3}$	...	g_{1,n_g-1}	g_{4,n_g}
$g_{4,1}$	$g_{1,2}$	$g_{1,3}$	...	g_{4,n_g-1}	g_{1,n_g}

$g_{1,1}$	$g_{1,2}$	$g_{1,3}$	...	g_{1,n_g-1}	g_{1,n_g}
$g_{2,1}$	$g_{2,2}$	$g_{2,3}$	...	g_{2,n_g-1}	g_{2,n_g}
$g_{3,1}$	$g_{3,2}$	$g_{3,3}$	...	g_{3,n_g-1}	g_{3,n_g}
$g_{4,1}$	$g_{4,2}$	$g_{4,3}$	...	g_{4,n_g-1}	g_{4,n_g}

Figure 4.19: Hyperparameter optimization with a genetic algorithm

Algorithm 6: Data augmentation strategy for encoded time series.

```
Input: $X_{encIn}$: encoded sensor signals
Input: $Y_{In}$: feature maps labels
Input: $n_X$: number of feature maps
Input: $n_F$: number of features
Input: $p_m$: percentage of feature maps that will be duplicated
Input: $d_L$: maximum number of rows that can be substituted by zeros
Output: $X_{encOut}$: augmented feature map array
Output: $Y_{Out}$: labels of augmented feature map array
Initialization
$X_{encOut} \leftarrow X_{encIn}$
$Y_{Out} \leftarrow Y_{In}$
$L_{min} \leftarrow argmin(L_{test})$
$n_{Mod} \leftarrow round(n_X \cdot p_m)$
$ind_{Mod} \leftarrow random\ choice(X_{encIn}, n_{Mod})$
for $i \leftarrow 1$ to $length(ind_{Mod})$ do
    $X_{temp} \leftarrow X_{encIn}[ind_{Mod}[i], :, :]$
    $f_k \leftarrow Generate\ random\ value(0, 1)$
    $f_k \leftarrow f_k \cdot d_L$
    $X_{temp}[1 : f_k] = zeros(fk, n_F)$
    $X_{encOut} \leftarrow append(X_{temp})$
    $Y_{out} \leftarrow append(Y_{in}[ind_{Mod}[i]])$
end
```

The problem of multiclass classification is solved by the *one-versus-one* approach. Therefore, $K(K-1)/2$ models are generated to identify the fault type, with K equivalent to the number of classes. The amount of models that can be stored in the embedded system depends on the size of each classifier and the available space in the memory unit. Moreover, it is important to consider that the evaluation of $K(K-1)/2$ models in real time operation can result in a high burden for the embedded system. Therefore, the Decision Directed Acyclic Graph (DDAG) architecture [68] is implemented to reduce the computing load. The basis of the DDAG approach is that after comparing two classes, all the models that consider the classification of the non-selected class are not evaluated anymore. Thereby, the number of computations to identify the class in the multiclass problem is reduced to only $K-1$.

Figure 4.20 depicts the approach based on multiclass SVM with DDAG architecture for the classification of multiple fault types (FT). First, the electric drive is monitored by N sensors. All relevant variables for each SVM model J are generated from the sensor signals and stored. Only the M most relevant features for each J classifier are feeded to the model, and the classification takes place. The output of this model is the fault class that affects the system. Fig. 4.20 depicts a model that is able to classify four different fault types. To this end, six classifiers are trained and three are evaluated during the on-board implementation.

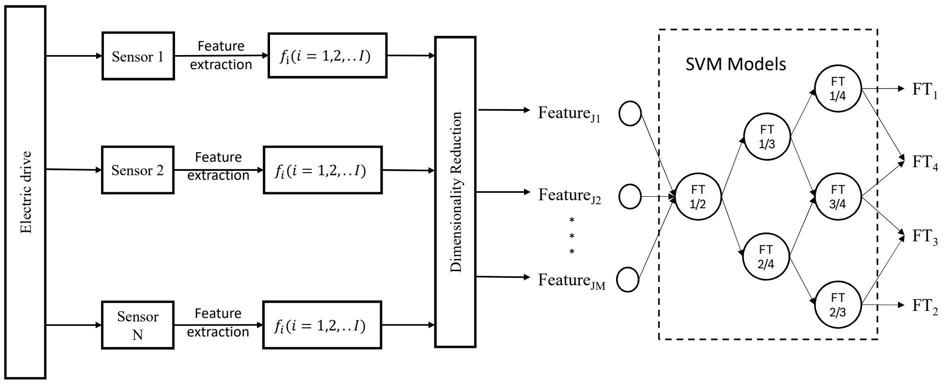


Figure 4.20: Fault isolation based on multiclass SVMs (Example with 6 SVM models for 4 fault types).

If there is a great amount of common fault types, which the multiclass SVM should separate, many SVM models would be neccesary. Nevertheless, each SVM model requires only few parameters to perform a prediction. The amount of parameters altogether would normally be lower than that of other type of architectures, such as neural networks. Moreover, the total number of possible fault types is usually just a low portion of the total amount of mechanical elements that build the electric drive. Thus, cases where over a dozen of different elements can cause a system malfunction have extremely low probability.

4.8.1 Fault diagnosis under varying settings

During normal operation, electric drives run with varying loads instead of a constant performance, as is the case in test benches. We assume that faults of certain components produce greater variance in the sensor signals in determined operation ranges. For example, a malfunction in a gear could produce variations on the sensor signals that might only be perceived when high torques are reached. Thus, if the information of the whole operation range is used for training the classifier, its performance would decrease.

A technique to find the optimal range for fault isolation is presented in Algorithm 7. First, the number of segments, in which the operational range is divided, is defined. Then, an array X_{range} is created with data that was gathered during the operation under the selected range. This data array is normalized with the z-score transformation. Afterwards, a predefined amount of time-based or frequency-based features are extracted from the raw signals. The dimensionality of the feature space is reduced with the SVM-RFE algorithm optimized with SA. To calculate the score, the energy obtained in the SVM-RFE approach and the number of data points, which constitute the data in the analyzed performance range, are considered. A data point refers to a measurement segment with length L. By using the factor γ/N_{points}, we assure that operating ranges with less than γ measurements are not chosen as optimal. The loop ends once the score calculation of all ranges is finished or a target score has been achieved. Finally, the range that achieved the lowest score is selected.

Algorithm 7 can be adapted to find the optimal range for regression tasks. To this

Algorithm 7: Search of optimal range for classification problems.

```
   Input: $X$: Data array
   Input: $R$: Set of ranges
   Input: $n$: Amount of ranges
   Input: $dt$: Amount of data points that are used to compute the statistical values
   Output: $R_{opt}$: Optimal range
   Output: $S_{opt}$: Optimal feature sub-set
1  Initialization
2  for $i \leftarrow 1$ to $n$ do
3      $X_{range} \leftarrow X \in R[i]$
4      $X_{stats} \leftarrow stats(X_{range}, dt)$
5      $X_{norm} \leftarrow normalize(X_{stats})$
6      $[S[i], E] \leftarrow SVM-RFE-SA(x_{norm})$
7      $Score[i] \leftarrow E + \gamma/N_{points}$
8  end
9  $f \leftarrow argmin(Score)$
10 $S_{opt} \leftarrow S[f]$
11 $R_{opt} \leftarrow R[f]$
```

end, the dimensionality reduction has to be performed with Algorithm 1. Afterwards, the regression model is trained with the reduced feature space. Since the GPR model requires less training time than the TCN or MTCN architectures, it is chosen to evaluate the different operation ranges. The energy E is then substituted by the fitness 4.10. Once an optimal range is found, a TCN or MTCN model can be trained with the data collected only during the operation of the electric drive in the selected range.

4.9 Towards on-board implementation

Although the developed algorithms focus on the creation of compact models that are suitable for an embedded implementation, there are still improvements that should take place prior to the on-board condition monitoring. This section introduces the process required to optimize the algorithms for an on-board implementation as well as an evaluation of when to use a traditional machine learning model for prognosis and when the deep learning architecture is a more suitable solution. But first, this section introduces how to estimate the size requirements of each model that should be deployed in an embedded system.

4.9.1 Models size requirements and computational complexity

Within this section, it is indicated how to estimate the size required in the Flash memory and in the RAM by each architecture. Furthermore, the estimation of the MAC operations is also given within this sub-section.

GPR

The execution of a prediction with a GPR model is an iterative process. Thus, only the kernel parameters, the input and the vector of the $i-th$ sample are required to be stored simultaneously in the RAM. The highest burden comes from the computation of the Euclidean distance (2.27) within the covariance function. The Euclidean distance can also be iteratively calculated as follows: $r_i^2 = x_n^2 - 2x_n x_{m_i} + x_{m_i}^2$, where x_n is the input and x_{m_i} is the $i-th$ sample within the dataset. The term x_n^2 is computed before starting the comparison with the samples of the GPR model to reduce the number of operations on the real time environment. The number of parameters that should be stored in the RAM due to the computation of r is $4 \cdot M$, where M is the amount of features. The value of 4 comes from the number of partial results that have to be stored plus the space for the final result. The parameters σ_n and σ_f of the covariance function are constant scalars. Finally, the prediction given by (2.37) requires to multiply the result of the covariance function with the parameter α_i, which is also another scalar. Table 4.2 summarizes the number of parameters that should be stored in the RAM for the RUL estimation with a GPR model.

Table 4.2: Number of parameters that have to be stored in the RAM for a prediction with a GPR model.

Process/ Variables	**Number of parameters**
Input x_n	$1 \cdot M$
Sample x_{m_i}	$1 \cdot M$
Euclidean distance r	$4 \cdot M$
Constant coefficients σ_n, σ_f, α_i	3
Output y_i	1
Total	$4 + 6 \cdot M$

The samples x_m and the kernel parameters are constant values that are stored in the Flash memory and are loaded to the RAM only when they are required. Moreover, the vector α is computed off-line and it is also stored as a constant vector in the Flash memory. This vector has a size of $\{1, N\}$. Thus, the number of parameters of a GPR model that are stored in the Flash memory is given by (4.12), where N is the number of samples.

$$par_{GPR} = N \cdot (M+1) + 3. \tag{4.12}$$

Finally, the amount of MAC operations is given by:

$$MAC_{GPR} = N \cdot (M+1). \tag{4.13}$$

SVM

Given a new input x_n, a prediction is given by (2.23). Considering a polynomial kernel of second degree, a new estimation is given by: $y_{pred_i} = \alpha_i \cdot y_i(x_n^2 - 2x_n x_{m_i} + x_{m_i}^2) + b$, where x_m are the samples that are support vectors. Similarly to the GPR, the prediction performed by an SVM is an iterative process. The amount of parameters stored in the RAM is very similar to that of the GPR given in Table 4.2, only that instead of storing 4 constant parameters, 2 constant values are saved, since a polynomial kernel, which has no hyperparameters, is used. Moreover, the number of parameters that are stored in the Flash memory is equivalent to:

$$par_{SVM} = N \cdot (M+1) + 1, \tag{4.14}$$

where N is the number of observations, which are support vectors. Finally, the amount of MAC operations is also given by 4.13, but with the difference that N is the number of observations that are support vectors and not the total amount of samples.

TCN and MTCN

The amount of parameters that have to be stored in the Flash memory for the deployment of both the TCN and MTCN models is equivalent to the total number of filter weights and biases of each network's layer. The number of parameters of a convolutional layer is given by (2.50), and of a fully connected (FC) layer is given by (2.43). To compute the amount of parameters that have to be stored in the RAM, a hardware architecture is considered, where six arrays with the in-between results of the layers are stored simultaneously. The capacity that has to be allocated in the embedded system for these arrays is given in Table 4.3.

The first array is required to save the input feature map. The slot required for the filters considers only one kernel, because the feature map is convolved with one filter at a time. Moreover, three feature maps generated by each convolution within the residual block plus the input feature map to the residual block are allocated in the memory. The output of the last residual block is flattened to connect it to a dense layer. The size of

Table 4.3: Number of parameters that have to be stored in the RAM for a prediction with the TCN and MTCN models.

Process/ Variables	**Number of parameters stored in RAM**
Input	$C \cdot W$
Filter	$C \cdot k_s$
Residual block	$4 \cdot k \cdot W$
Flatten	$n_{flatten} \cdot k$
Dense	$(n_{flatten} \cdot k + 1)n_{dense}$
Output y_i	$n_{dense} + 1$
Total TCN	$C(W + k_s) + k[4 \cdot W + n_{flatten}(1 + n_{dense})] + 2 \cdot n_{dense} + 1$
Total MTCN	$C(W + k_s) + k[4 \cdot W + n_{flatten}(1 + 2 \cdot n_{dense})] + 3 \cdot n_{dense} + 1$

the resulting vector is determined by the number of time steps that are flattened and the number of filters. This, because the size of the feature map remains constant after the first 1D convolution, where the number of channels C is substituted by the amount of filters k. The space neccesary to store the operations of the dense layer depends on the size of the flattened vector and the number of neurons of the FC layer. As for the MTCN, only the partial results of the FC layers of both paths have to be stored simultaneously.

The amount of MAC operations of the convolutional layers is given by (2.51). In the input layer of the TCN and MTCN architectures, k_s is equal to one. The output of this layer has a size of $\{W, k\}$, which remains constant through all the residual blocks (RB). Thus, the convolutional layers within the blocks are multiplied by the factor k^2. The number of MAC operations of a single residual block is multiplied by the amount of RBs, which is determined by the coefficient dil_{RL}. The number of MAC operations of each architecture is summarized in Table 4.4.

Table 4.4: MAC operations of TCN and MTCN architectures.

	Process/ Variables	**MAC operations**
	Input convolution (M_0)	$1 \cdot k \cdot C_{in} \cdot W$
RB	Dilated causal convolutions (M_{1a})	$k_s \cdot k^2 \cdot W$
	1D convolution (M_{1b})	$1 \cdot k^2 \cdot W$
	Total RBs (M_1)	$dil_{RL}(2 \cdot M_{1a} + M_{1b})$
	Dense Layer (M_2)	$n_{flatten} \cdot k \cdot n_{dense}$
	Total TCN	$M_0 + M_1 + M_2$
	Total MTCN	$2(M_0 + M_1 + M_2 + n_{dense})$

4.9.2 Model optimization for embedded deployment

Model size compression

We mainly intend to reduce the size of a learning algorithm to enable its deployment into an embedded system or to release hardware resources for other applications. Within this chapter, three different learning algorithms have been introduced for fault diagnosis and failure forecasting. Depending on the chosen algorithms, there is a procedure that takes place to reduce the size of the model by simultaneously mantaining a bearable performance. Figure 4.21 depicts an overview of the size optimization methods for each learning model.

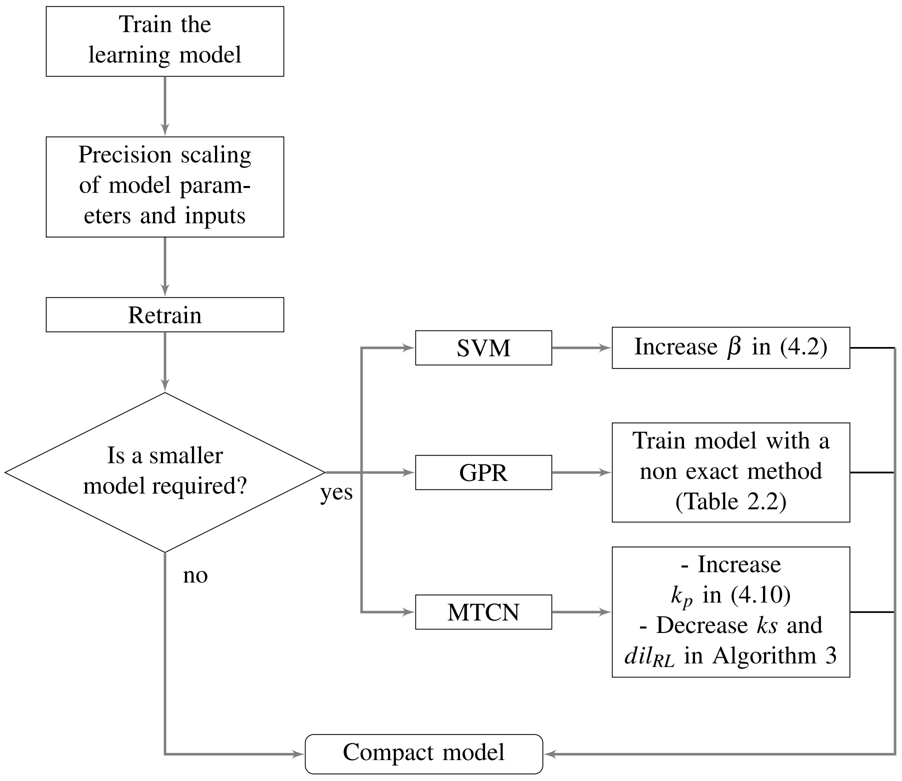

Train the learning model
Precision scaling of model parameters and inputs
Retrain
Is a smaller model required?
yes
no
SVM
GPR
MTCN
Increase β in (4.2)
Train model with a non exact method (Table 2.2)
- Increase k_p in (4.10)
- Decrease ks and dil_{RL} in Algorithm 3
Compact model

Figure 4.21: Model size reduction for the diagnosis and prognosis algorithms covered within the proposed approach.

First, the chosen learning algorithm is trained without implementing a strategy for size reduction, in order to prove the suitability of said model to solve the desired task. Once the model reaches an acceptable performance, the precision of parameters and the input signals is scaled from 32 to 16-bit floating point. Thus, the model size can be reduced up to the half. The efficiency of the architecture is evaluated and if the loss on performance is aceptable, the model can be deployed. Otherwise, the model is retrained with the 16-bit inputs. If the model has an aceptable performance, but it is intended to reduce its size more, then a specific procedure that depends on the selected learning algorithm takes place. To reduce the size of the SVM classifier, the factor β is increased in (4.2). Thus, during the dimensionality reduction approach, the feature subspaces with less variables are preferred over more accurate models with more features. From (2.23) we know that an estimation depends on the lagrange multipliers α, the values of support vectors x_n and their labels y_n. The size of α and y_n is given by $\{p, 1\}$, where p is the number of samples that are support vectors. It is possible that with less variables, a lower amount of support vectors is found, reducing thus the classifiers size. Nevertheless, the greatest compresion is derived from the change on size of x_n, which is equivalent to $\{p, f\}$, where f is the amount of features. Moreover, less variables means that less space has to be dedicated to store the inputs of the SVM classifier.

A new prediction with a GPR model depends on the kernel parameters and the previous observations (2.37). If the log marginal likelihood (2.38) is maximized using an exact method, the GPR requires to store the complete training dataset into the embedded system to perform a new estimation. Thus, the method to reduce the models size is to reduce the training dataset volume. This can be performed by using an approximation method of Table 2.2. These methods select determined samples to train the GPR, reducing thus the training time and the model size.

Finally, the MTCN model can be compressed by increasing the factor k_p in (4.10) during the hyperparameter optimization. This constant penalizes networks with a greater number of parameters. With a higher value for k_p, the GA searches a solution that is not the most accurate but has an acceptable performance with a smaller architecture size. Moreover, within the automatic HP tuning, the maximum possible values of the kernel size and dilation rate can be lowered prior to the creation of the initial population (Algorithm 3). In this way, the GA searches a solution that has a lower maximum receptive field. Therefore, the architecture has less depth and the feature maps are smaller.

Reduction of computational complexity

Not only the size of the models have an influence on their suitability to be ported in determined embedded systems, but also the number of MAC operations that they require to make a new prediction plays an important role. On one hand, machine learning models are very compact and require just a few amount of MAC operations compared to deep learning architectures. Thus, no technique is suggested to reduce their MAC operations within this work. On the other hand, the MTCN and TCN can still require hundreds of thousands of MAC operations to perform a new estimation, even though they consist of just a couple of thousands of parameters.

A way to reduce the amount of MAC operations of the MTCN is by reducing the value of the coefficient dil_{RL}. This would generate less residual blocks in each path of the architecture. Thus, the decrease of the MAC operations would be considerable and might make the model suitable for a determined embedded application. Moreover, a pooling layer can be connected after the first convolution to reduce the width of the feature map that is given as an input to the residual blocks. This would generate a reduction of the MAC operations of the total residual blocks (M_1) by a factor of p, which is the pool size. Other methods for model size compression and reduction of computational complexity of DL models, are described in Section 7.1.

4.9.3 Comparison of learning models for online failure prognosis

Throughout this chapter, three different regression algorithms that solve the task of the RUL estimation have been detailed. This section highlights the advantages of every model and provides a short guideline of when each model is the most appropiate solution for a given application. The characteristics of the regression algorithms that are relevant for our approach are reported in Table 4.5.

Table 4.5: Comparison of prognosis algorithms.

Model	**Performance**	**Memory required**		**Computational Complexity**
		Flash	**RAM**	
GPR	Medium	**Low**	**Very low**	**Very low**
TCN	High	Medium-High	Medium	Low
MTCN	**Very high**	Medium	Medium	Low

First, the GPR is a very compact model with the least amount of parameters. The GPR is ideal to quickly test the feasibility to perform the RUL estimation in the embedded system. Nevertheless, its performance is clearly lower than most deep learning architectures. Thus, it could cause late predictions or false alarms. This algorithm is useful to have an overview of the RUL online, but decisions taken based only on its predictions are not advised. The TCN improves greatly the accuracy of the GPR, but at a cost of larger model size and computational complexity. The MTCN increases the generalization capability of the TCN with a similar model size and often, the MTCN results in a more compact architecture. The disadvantage is that the amount of MAC operations of the MTCN is two times larger than a TCN with the same hyperparameters. The RUL estimation is a task that doesn't have to be performed continuously. In the case that the monitored electric drive is used for a vehicle, the RUL prediction can take place once during a ride. Moreover, not all computations must take place at once, but partial results can be generated while other functions with more priority are in standstill, for example during a red light. Therefore, the slightly higher number of MAC operations required by the MTCN wouldn't hinder its deployment for an on-board monitoring. The proposed MTCN enables accurate predictions online that can be used to take a decision about the optimal time at which the repairs or component replacement should be performed.

The experimental study that is presented in the following chapter demonstrates the suitability of the developed methodology to identify, forecast and diagnose failures of mechanical elements of a pedelec's electric drive. Moreover, a second study compares the RUL estimation approach against other related algorithms using a publicly available database.

Chapter 5

Case study: Condition Monitoring of Pedelec Drive Units

This chapter introduces two case studies. The first one deals with the failure forecasting of electric drives for pedelecs. To this end a database that consists of measurements of two types of electric drives in a validation test known as endurance test was used. In the second case study, a fault diagnosis of drive units (DU) for electric bicycles was performed. In this case, the data was gathered directly in field measurements. The results presented in this chapter have been published in [101] and in [61].

The generation and training of the GPR and SVM models was performed with Matlab R2019b and a 32 GB RAM. On the other hand, the training of the MTCN and TCN architectures was performed with an NVIDIA Quadro P2000 with 5 GB, CUDA 10.2 and Tensorflow 2.1 [2].

5.1 Description of the pedelec drive units

Electric bicycles or e-bikes can be definied as ”bicycles that are similar in geometry to human-powered bicycles but have a small electric motor that provides pedal assistance and allows riders to accelerate, climb hills, and overcome wind resistance more easily than manually powered bikes” [56]. The term pedelec is frequently used as a synonym of e-bike. Nevertheless, there is a difference between both concepts. In an e-bike the driver might only press a button, so the motor is activated. On the other hand, the pedelecs provide support only when the driver applies force to the pedal. This is a crucial difference, since the support given by the drive unit of pedelecs depends directly on the amount of torque that the driver applies.

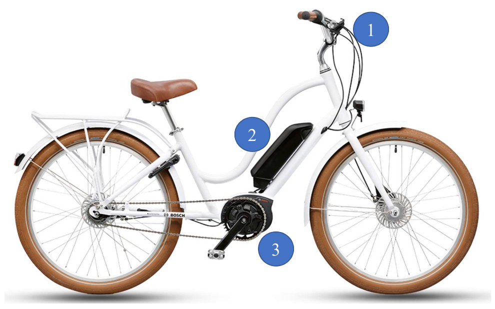
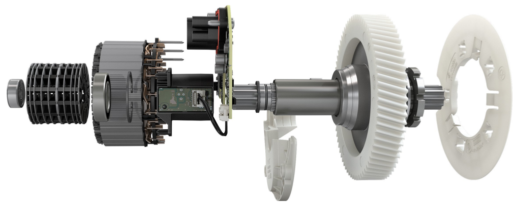


Figure 5.1: Pedelec system overview.

Figure 5.1 depicts the overview of a pedelec and its three main components: the human machine interface (1), the battery (2) and the drive unit (3). There are three main drive types: front hub, rear hub and mid-drive [3]. For the present case study two drive units of the mid-drive type are analyzed. The first one has a rotating electric motor and one gear stage, which together deliver a maximum torque of 40 Nm [75]. An overview of this system is depicted in Fig 5.2a. The second drive has a similar configuration, but with three gear stages, which increases the maximum delivered torque up to 85 Nm [75]. Figure 5.2b shows the overview of the second drive unit type. Throughout this case study, the drive unit with one gear stage is named as Type-A DU and the electric drive with three gear stages as Type-B DU.


(a) Overview of Type-A DU.

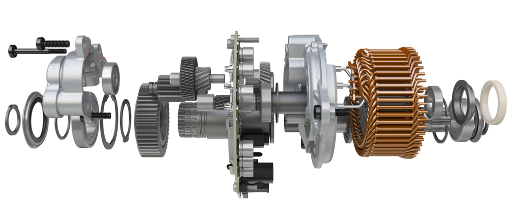

(b) Overview of Type-B DU.

Figure 5.2: Overview of two drive unit types [75].

5.2 Failure prognosis of drive units for pedelecs

The first case study has the objective to test the efficiency and flexibility of the proposed failure forecasting approach with two different types of electric drives.

5.2.1 Database description

The dataset consists of the data gathered within the endurance tests of 74 samples of the Type-A DU and 64 of the Type-B DU. In an endurance test, the functionality of the DUs is assessed under extreme operating conditions during a very long time period. These operating settings do not represent the common operation of the systems. Since this test takes place during the product development phase, the tested prototypes differ from each other; for example in the materials, suppliers, manufacturing process, etc. The main objective of the test is to validate that a certain design constructed with determined materials and

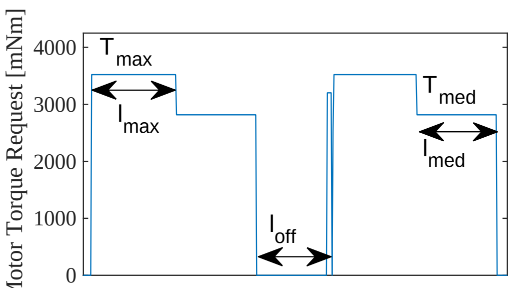


Figure 5.3: Profile of the endurance test for pedelec drive units.

with a specific manufacturing process fulfills the minimum endurance specified by a norm, before it is launched to the market.

The profile of the endurance test is depicted in Figure 5.3 and is as follows: First, the DU runs with a maximum load T_{max} for a period l_{max}. Then, the drive is operated with a lower torque T_{med} for a shorter period l_{med}. Finally, the electic drive is turned off for a period l_{off}. These three segments are repeated throughout the test, which is concluded after $n - hours$ have elapsed or when the system breaks down.

Data cleansing

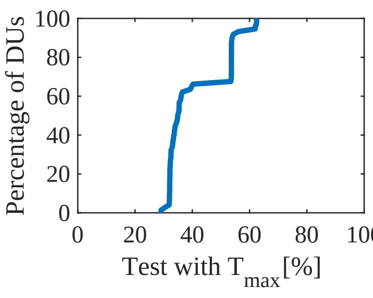
Within each test, over 50 signals were collected with a sampling time between one and sixty seconds. These values corresponded to test parameters, drive unit sensors, and external sensors. Prior to the implementation of the approach, a data cleansing stage took place. Here, only those signals that were generated from the DU internal sensors were kept. Then, all the variables that were constant or consisted of non numerical data were removed. In this way, the data was structured as indicated in Section 4.1.2. The database was reduced to only 8 variables, which are enlisted in Table 5.1.

Table 5.1: Variables of the endurance test database after data cleansing.

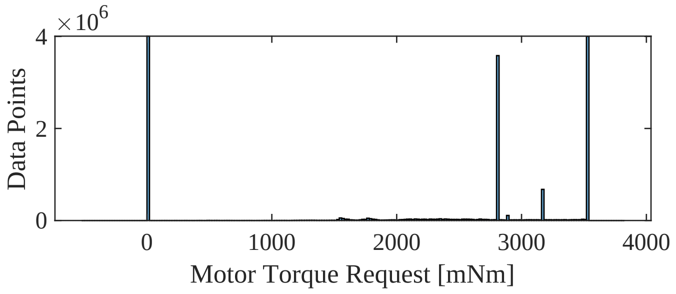
Type	Signal	Abbrev.
ID	Drive unit number	DUN
Time	Test running cycle	Cycle
Operating settings (X_{set})	Motor torque request	MTR
	Chamber temperature	Temp
Drive unit signals (X_{sig})	Motor torque present	MTP
	Motor angular speed	nMotor
	Driver torque	Tin
	Torque at the output shaft	Tout

The analysis of the motor torque within all experiments let us identify two core elements of the tests: In general, the DUs operated mostly under the maximum load and the value of T_{max} and T_{med} varied among the samples. As seen in Figure 5.4a, almost all DUs ran at least one third of the complete test with maximum load. Furthermore, 35% of the systems were operated more than the half of the test with T_{max}. The second most frequent state was with the motor switched off, as seen in Figure 5.4b. The data gathered during the motor standstill was not relevant and the information generated while the DU operated with T_{med} was minimal. Thus, only the data collected when the DU was operated with maximum load was considered for this analysis.

Figure 5.5a depicts the angular speed of a drive unit, which was collected while the system was required to output its maximum load. In this chart, it is seen how the signal had a great amount of peaks, which difficulted the identification of a trend with the time. These fluctuations in angular speed caused also peaks in other signals. Therefore, a saturation filter was applied to minimize the effect of these speed variations. The filtered signal is also shown in Figure 5.5a. Moreover, the maximum torque varies among the tests. This can be seen in Figure 5.4b, where three peaks that correspond to the load T_{max}



(a) Test percentage with T_{max}

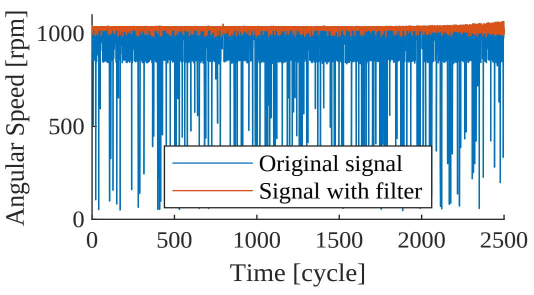


(b) Distribution of MTR.

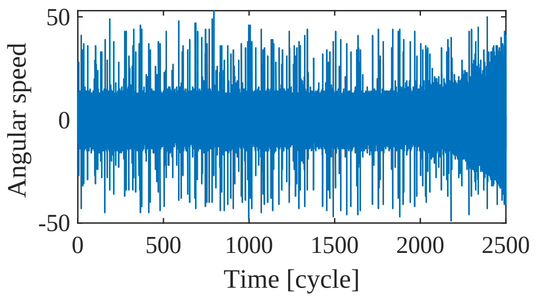
Figure 5.4: Statistics of Motor Torque Request of all Type-A DU tests.

within all tests are depicted. In order to have similar amplitudes, the signals in X_{sig} were centered around zero by computing the difference of the adjacent elements. Figure 5.5b shows the angular speed centered around zero. These pre-filtered and centered around zero signals constructed the dataset, which worked as input to the methodology for failure prognosis introduced in Chapter 4 .

The time vector is given in cycles. A cycle is considered as the segment, in which the drive unit runs with a load T_{max} and has a duration of l_{max} (see Fig. 5.3).



(a) Angular Speed under T_{max}.



(b) Angular Speed centered around zero.

Figure 5.5: Angular speed collected only during the operation with T_{max}.

5.2.2 Generation and preprocessing of relevant features

There were in average 1.5 million data points for each test. This included the data of the 8 variables from Table 5.1. Traditional machine learning algorithms require a feature extraction process, in order to process this amount of data, since they are not able to extract the features from long time series by themselves. On the other hand, deep learning models enable an automated feature engineering even with this vast dataset. Nevertheless, this might require a much bigger DL model and a feature map with a greater width. These two factors can difficult the implementation of the model in an embedded system. Therefore, it was decided that for this large dataset, a manual feature extraction takes place prior to the training of both learning algorithms. Moreover, to train the GPR model, a feature selection step was performed to identify the most important prognostics parameters. As for the case of the MTCN, the input feature maps were built with all the generated features and the operating settings.

Feature extraction from the raw signals

Though the database consists of long time series, the data was collected with sampling times over one second. Therefore, only the following time-based features were extracted from each DU signal: mean (2.1), standard deviation (2.4), peak to peak (2.11), kurtosis (2.6) and skewness (2.5). For the computation of these functions, a set of 20 samples of each time series were considered. In Figure 5.6, four of these statistical values extracted from the angular speed signal of the 10th DU sample are depicted. As seen in this chart, the standard deviation and peak to peak provide the clearest overview about how the condition of the DU worsens with the time until the failure of an element arises. By doing this data transformation, the amount of samples was reduced by a factor of 20. The resulting data array X_{stats} was composed of 20 statistical functions instead of 4 raw signals like X_{sig}.

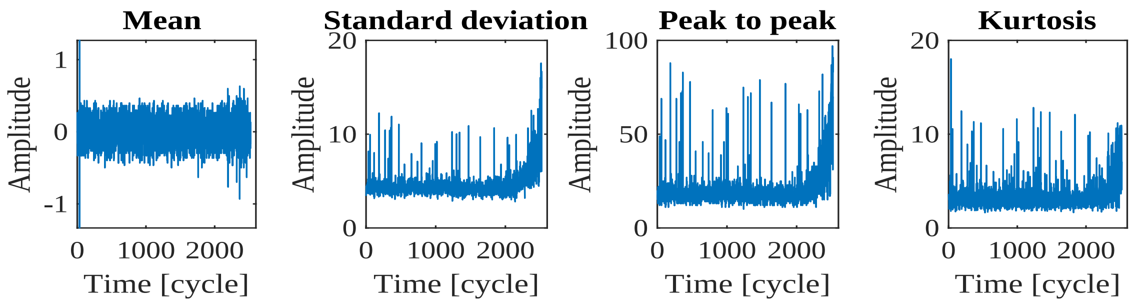
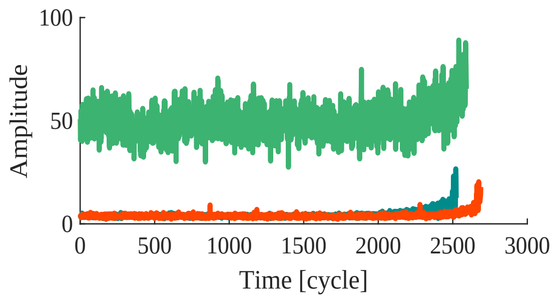


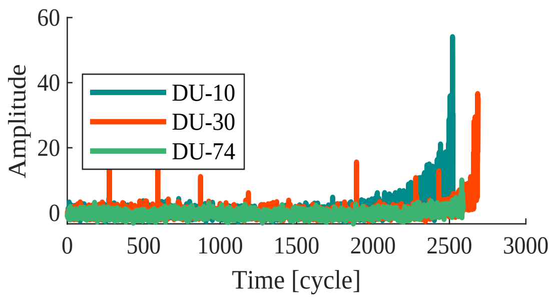
Figure 5.6: Time based features of the angular speed (DU Sample 10).

Preprocessing of the extracted features

The data array X_{stats} was standardized by means of equation 4.1. In chapter 4, it was defined that the reference range should be selected in a scope within 5-10% of the expected life of the systems. Since the expected lifetime of the DUs in this special test is 3000 cycles, the reference range was chosen as 150 to 250 cycles. Fig. 5.7a displays the MTP standard deviation of three different drive units. It is seen that this variable was not centered to a similar value, and there was a shift within its amplitude among different samples. In Fig. 5.7b, the normalized standard deviation of MTP is shown. After the normalization with equation 4.1, this extracted feature was centered around zero during the operation of the DUs without any anomaly and its amplitude raised with an increasing damage of an element. The normalized array was stored in the variabe X_{norm}.



(a) Standard deviation of MTP.

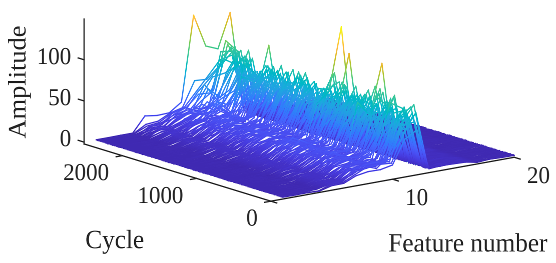


(b) Normalized standard deviation of MTP.

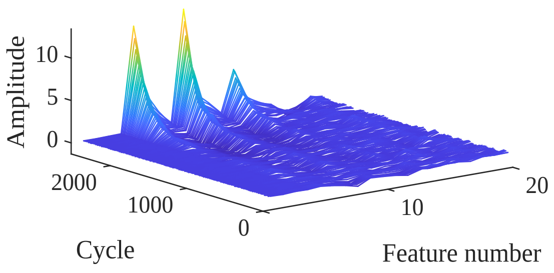
Figure 5.7: Signal preprocessing stage.

Moreover, as seen in Figures 5.6 and 5.7, the statistical functions of the raw signals were affected by random noise. In order to improve the performance of the GPR model, the features were smoothed with a one sided mean moving average filter (2.15) with a window equivalent to five. Thereby, the array X_{smooth} was created. On the other hand, the proposed MTCN model has a better performance when trained with noisy inputs. Thus, the signal smoothing was restricted only for the application with the GPR model.

Figure 5.8a depicts all the extracted features of a single DU without normalization and Fig. 5.8b shows these variables after the preprocessing stage has been performed. In this chart, it is also clear how the features have amplitudes close to zero during the initial cycles. Some of them have a progressing increase of the amplitude until the system failure artises. Other features remain almost constant throughout the test. In the following subsection, the selection of the relevant prognostics parameters, which were used to label the data and train the GPR model, takes place.



(a) Features without preprocessing.



(b) Preprocessed features.

Figure 5.8: Preprocessed signals of the drive unit sample 10.

Selection of the relevant prognostic parameters

The selection of the relevant features for the RUL estimation task was performed according to Algorithm 1. The fitness quantities of each variable of X_{smooth} are depicted in Figure 5.9. The standard deviation and peak to peak variables achieved the hightest fitness, because they show a clearer trend of the degradation of a system with the time. A value of $P_s = 0.5$ was chosen within Algorithm 1, i.e. the variables that made 50% of the cumulative sum were selected to form the new data array X_{red}. The reduced array consisted of the six variables with the highest fitness, which were: *std-MTP, std-nMotor, std-Tin, peak-MTP, peak-nMotor* and *K-Tout*.

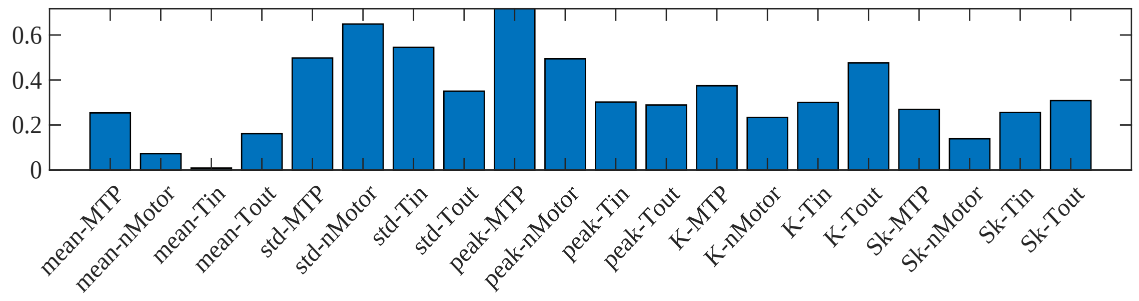


Figure 5.9: Fitness that indicated the suitability of each variable as prognostic parameter.

5.2.3 Data labeling

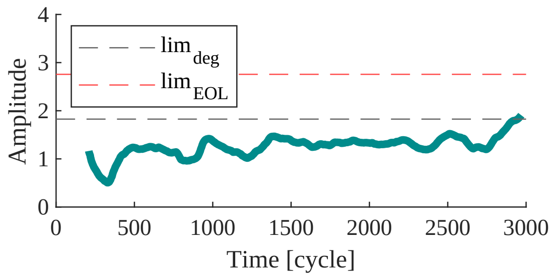
The endurance tests have mostly a fixed duration and there are only a few run-to-failure experiments. Moreover, some tests concluded before the desired duration is reached. Since the drive units had a great variance on their final condition at the end of the endurance test, the last measurement could be considered as the end of useful life as with the C-MAPPS database [83]. The procedure introduced in Section 4.5 was performed to determine the end of useful life of each system. Thereby, the data could be labeled.

Health index computation and threshold determination

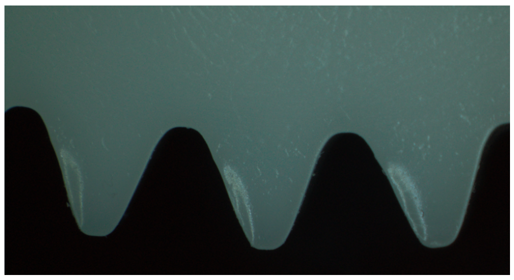
The first step in the data labeling procedure was the transformation of the multidimensional feature space X_{red} into a one dimensional metric known as health index (HI), which assesses the condition of the system. To this end, the Mahalanobis distance (2.10) was implemented. The reference distribution X_{ref} was selected as the data of all drive units that was collected within the cycles $150-300$. Then, the limit values were computed with equation (4.5). The degradation limit (lim_{deg}) was equivalent to 1.82 and the failure threshold (lim_{EOL}) to 2.75. The cycle at which the degradation of each DU began, was stored in ind_{deg}. Furthermore, the time of the failure occurrence was saved in ind_{EOl}. When the HI of a certain DU has reached lim_{EOL}, the probability that it suffers a sudden malfunction is too high. Within this case study, it was intended to predict how many cycles the systems can still operate before they arrive to this state. After the degradation and failure limits were estimated, the systems were classified in three groups, as indicated in Section 4.5.4. There were in total 8 class A DUs, 32 class B DUs and 34 electric drives that reached the end of useful life within the test, i.e. they were categorized into the C class.

Validation of the selected limits

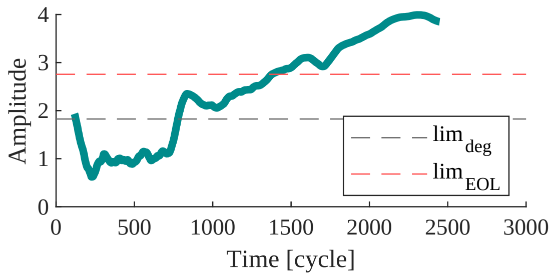
After the conclusion of each endurance test, the DUs go through a mechanical analysis. There, each element of the electric drive is examined to determine its degradation level. Through this inspection it is possible to determine the failure cause of the DUs that are not able to reach the desired test duration. The information collected wihtin these inspections was used to validate the thresholds and how the systems were categorized.



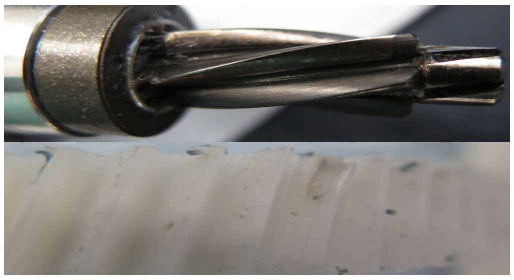
(a) HI of Class B DU (sample 33).



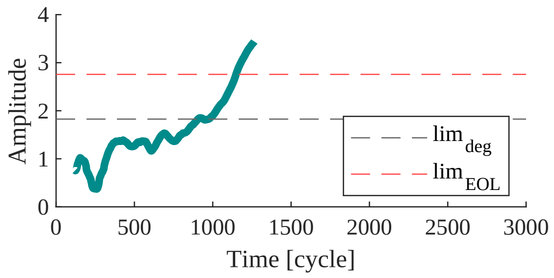
(b) Mechanical analysis of DU sample 33.



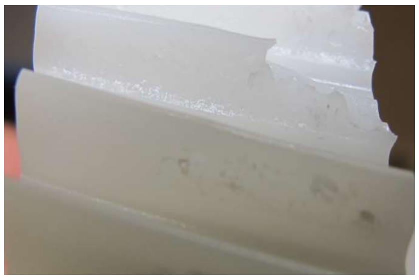
(c) HI of Class C DU (sample 49).



(d) Mechanical analysis of DU sample 49.



(e) HI of Class C DU (sample 47).



(f) Mechanical analysis of DU sample 47.

Figure 5.10: Validation of selected thresholds by means of mechanical analysis.

Figure 5.10 depicts the HIs of three different DUs and their respective mechanical analysis. First, Figure 5.10a shows the curve of the DU sample 33, which is categorized into the B group. This system could complete the desired test duration. Actually, the

HI exceeded the first threshold only at the end of the endurance test. The mechanical analysis indicated that only the gear had some traces of wear, which could only be perceived through a microscope, as seen in Figure 5.10b. This damage did not affect the operation of the drive. Figure 5.10c displays the HI of a class C DU. This system could not complete the entire test due to extensive wear traces present on the gear and on the rotor shaft, as seen in Figure 5.10d. Finally, the HI of sample 47 is shown in Figure 5.10e. Just around 100 cycles after the EOL limit was exceeded, a gear tooth broke, as depicted in Figure 5.10f.

Through the mechanical inspections it was possible to validate the following points:

- The constructed health index enables a correct monitoring of the system.
- The selected thresholds can correctly indicate when a DU has reached its end of useful life.
- After the EOL limit is reached, the probability of a sudden failure is very high.
- The proposed approach is a fast and efficient method to label the data for the RUL estimation task.

Finally, as seen in Figure 5.10, the test did not stop once the system had a considerable damage but only when the DU was not able to continue its operation. This proposed data labeling approach is also suitable to be applied as a condition monitoring system. Therefore, the tests could be stopped once an element is already damaged and thus, spare test-time, which can be used to start with the following experiment.

Generation of the dataset to train and validate the regression algorithms

The last step prior to the training of the learning algorithms is the definition of the data that constitutes the training, validation and test datasets with their corresponding labels. The label of each data array collected at a sample i is given by a piecewise linear target function (4.7), where the limit $maxRUL$ is determined by (4.8). The labels were stored in the target vector Y. Furthermore, as indicated in Section 4.5, mechanical systems have long periods operating without any anomaly and then have an exponential condition worsening that causes a malfunction. Due to this fact, there was a data imbalance, where most of the data was labeled with $maxRUL$ and there was a smaller percentage of data with a different label. Thus, the data was undersampled following the strategy introduced in Section 4.5.4. First, only the DUs categorized into the C class were considered for the final data set. Then, only the data collected within $ind_{deg} - kr$ and ind_{EOL} was considered to construct the final dataset, where kr was selected as 500. The final dataset was separated in training, validation and test subsets as indicated in table 5.2. This table indicates the rate, with which the data was separated into train, validation and test subsets. The GPR models require only a train and test subsets, since there are not intermediate evaluations of the model, which could enable the selection of a model with the lowest validation loss. Within this table, X_{red_U} refers to the undersampled X_{red} array; X_{norm_U} to the undersampled X_{norm} array; and Y_U to the undersampled target vector. The feature space size of X_{norm_U} is 22 because this array contains the 20 statistical quantities and the 2 operational settings.

Table 5.2: Properties of the Type-A DU dataset used to train the learning algorithms.

Parameter	**GPR**	**MTCN**
Dataset	X_{red_U}, Y_U	X_{norm_U}, Y_U
Dataset size	$\{36050, 5\}, \{36050, 1\}$	$\{36177, 22\}, \{36177, 1\}$
Size train set	85[%], 29 DUs	75[%], 27 DUs
Size validation set	-	15[%], 5 DUs
Size test test	15[%], 5 DUs	5[%], 2 DUs
maxRUL	300	

5.2.4 Failure forecasting

RUL estimation with a GPR model

First, the train data X_{red_U} was scaled in the range $[0, 1]$. The scaled data was the input to the GPR training approach depicted in Figure 4.12. An exponential kernel (Table 2.1) was selected as covariance function. Then, two models were created. The first one was trained with the complete training dataset and the second model was trained with the FITC [69] approximation method to reduce its size. The output of the GPR models was smoothed with a one-sided mean moving average with a window equal to ten.

Figure 5.11 displays the RUL predictions of two different samples performed by both GPR models. In the first chart, a sample that belongs to the training dataset is shown. The second graph depicts a sample of the test dataset. As seen in Figure 5.11, the GPR models perform an efficient RUL estimation for the sample belonging to the training

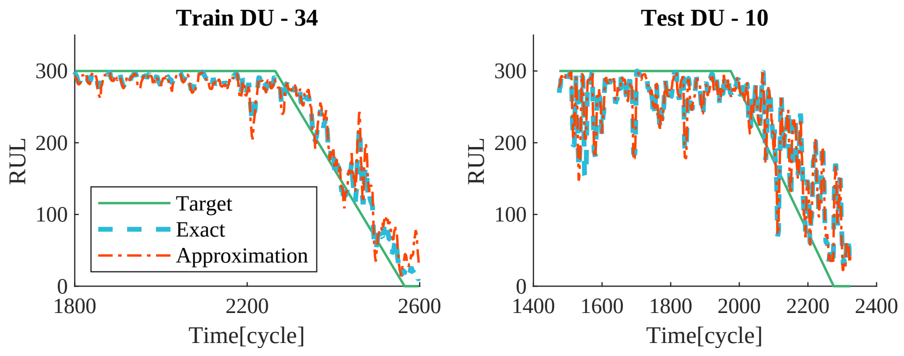


Figure 5.11: RUL predictions with a GPR model for two different DU samples.

dataset. However, they have difficulties to accurately estimate the RUL of the electric drives of the test dataset, mainly when the EOL is close (< 50 cycles). As seen in both charts, the estimations performed by both the exact and approximated models are very similar. Nonetheless, the approximated model has a compression rate of over 13 times, as indicated in Table 5.4. In Section 5.2.5, a comparison between the GPR models and the TCN and MTCN architectures in terms of generalization capability, model size and computational complexity is given.

RUL estimation with a TCN and MTCN architectures

First, a vanilla TCN architecture was built and then, its optimal hyperparameters were selected with the approach introduced in Section 4.7.3. Prior to the HP optimization, the array X_{normU} was scaled in the range $[0,1]$. The limits l_{low} and l_{up}, which are required to generate the random populations (Algorithm 3), are reported in Table 6.3. Each population P consisted of 16 individuals. Each indivual had 9 genes and was trained for 30 epochs. The fitness of every individual is given by equation (4.10). After the complete population was evaluated, the parents were selected by means of Algorithm 4 and the crossover was performed as indicated in Section 4.7.3. A restricted amount of genes of the resulting population P_{new} were *mutated* with the process described in Algorithm 5. The inputs for this algorithm were the following: $mut_{rate} = 0.3$, $\Delta mut_{rate} = 0.95$ and $g_{pop} = 144$. The termination condition was given after all the desired populations have been evaluated. In this case, 50 populations were assessed for the TCN model. The optimized HP search took in total 27.3 hours to evaluate the 800 models. In Fig. 5.12, the distribution of the fitness values as well as the median of each evaluated population are displayed. For clarity of the figure, the outliers are not displayed. Within this chart, it is possible to identify the downtrend of the fitness values. The optimal individual within this search is found in the population 47. The genes of this optimized solution are displayed in Table 5.3.

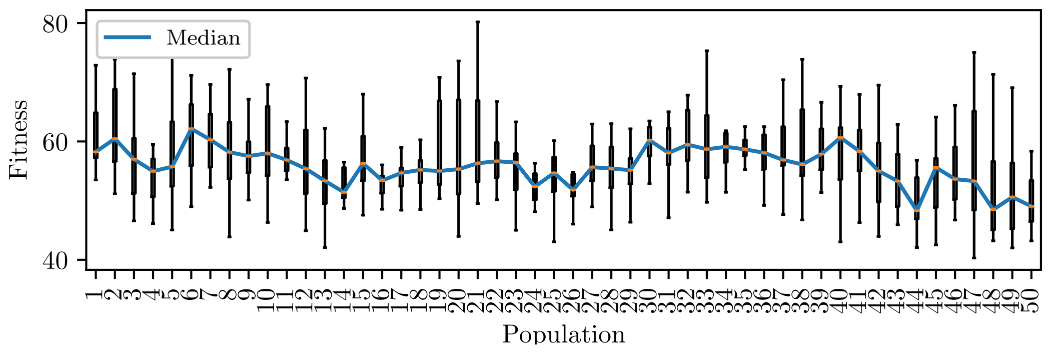


Figure 5.12: Box plot of the fitness at each epoch of the hyperparameter tuning approach with a GA.

The search of the optimized HPs for the MTCN model was started with a population that consisted of 8 random generated individuals and the 8 individuals with the best results of the HP search corresponding to the TCN. In this case, each individual had 10 genes, because the standard deviation of the Gaussian noise g_N is also considered as a HP. Only 15 populations were evaluated for this search. The optimized hyperparameters for the MTCN model are also displayed in Table 5.3.

Table 5.3: Optimized hyperparameters obtained with a GA for the Type-A DUs.

	ks	dil_{RL}	nb_{filt}	w_t	dr	η	bs	n_{dense}	$n_{flatten}$	g_N
TCN	5	2^4	8	3	0.2	1.7e-3	30	85	10	-
MTCN	5	2^4	8	3	0.2	1.7e-3	30	10	10	0.1

Finally, the data augmentation strategy described in Algorithm 6 was implemented, and both optimized models were retrained for 200 epochs to further improve their accuracy and generalization capability. Figure 5.13 depicts the RUL estimation of three different drive units. Each DU belongs to a subset, namely training, validation and test datasets. Contrary to the GPR models, it was not neccesary to smooth the output of the TCN and MTCN architectures to achieve understandable results. Futhermore, these models have a better efficiency when the systems are close to the EOL than the GPR. In these charts, it can be seen how the MTCN delivers an output that is closer to the target function than the prediction generated by the TCN model. This can mainly be noticed in the curves of the DUs that belong to the validation and test datasets. The following subsection gives a numerical comparison among the TCN and MTCN models.

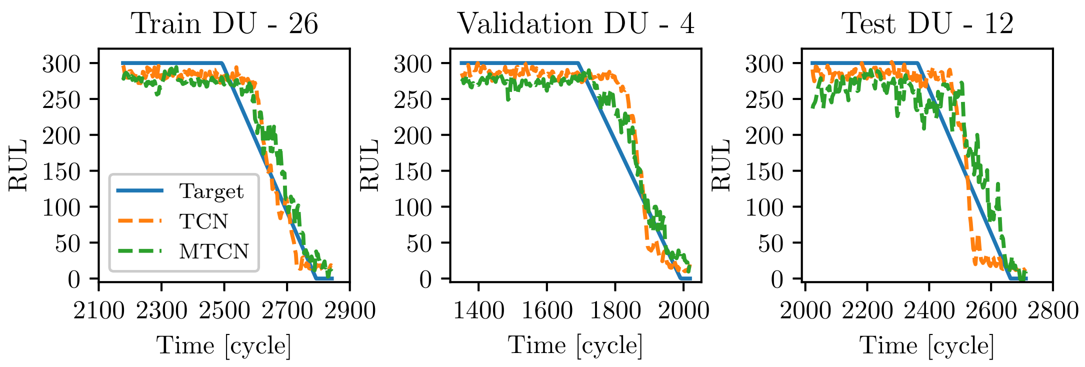


Figure 5.13: RUL predictions with TCN and MTCN models for 3 different Type-A DU samples.

5.2.5 Results and models comparison

The RMSE (4.9) and the cumulative relative accuracy (CRA) metrics are used in the evaluation of the models. The CRA is defined as a normalized weighted sum of relative accuracies at specific time instances [85] and it is given by:

$$CRA = \frac{100}{|p|} \sum_{i \in p} w(i) RA, \tag{5.1}$$

where p is the set of all the indexes where a prediction is performed, $|p|$ is the cardinality of the set and $w(i)$ is a weighting function depending on the RUL. This weighting function is normally selected so that predictions closer to the EOL have a greater importance in the metric computation. In this approach, we select the indexes $p = [300, 250, 200, 150, 100, 50, 1]$, which are n cycles before the EOL and a linear weighting function $w(r(i)) = a_1 p + a_2$, where $a_1 = -1/300$ and $a_2 = 1$, so that for predictions close to the EOL $w(i) = 1$ and for the indexes close to maxRUL $w(i) = 0.8$. RA is the relative accuracy, which is defined as follows:

$$RA = 1 - \frac{|y_{predicted_i} - y_{target_i}|}{y_{target}}. \tag{5.2}$$

Reproducibility of the models

To determine the reproducibility of the models, each architecture was trained 20 times. In every iteration, the training, validation and test datasets were changed. The TCN and MTCN were trained for 50 epochs. Figure 5.14 depicts the RMSE and CRA values of the four evaluated models. We can directly observe that the variance in the results of the deep learning architectures is much greater than of the GPR models. This implies that this traditional machine learning approach is more precise than the evaluated DL architectures.

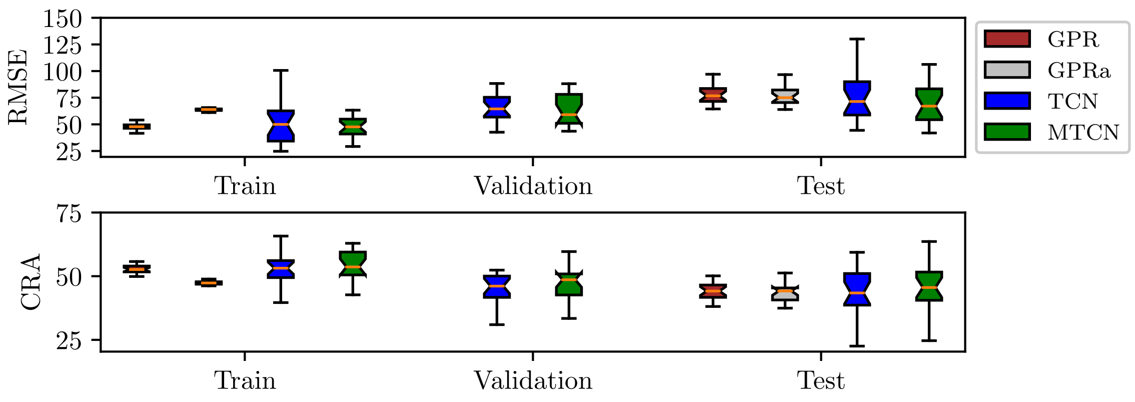


Figure 5.14: Models reproducibility evaluation

In general, the GPR trained with the exact method achieved a better performance for the train set. However, the generalization capability of the approximated method matched the performance of the exact GPR and even some GPR approximated models achieved a slightly better result in the test set. Moreover, the deep learning architectures outperformed the GPR models. Particularly, the proposed MTCN had the best performance for the validation and test subsets among the four evaluated models.

Performance evaluation

Table 5.4 summarizes the performance of the four evaluated models. The displayed RMSE, CRA and training time results are the median values obtained during the evaluation of the models reproducibility. The MTCN outperformed the other three models in both metrics and for each data subset. The exact GPR and the TCN had a similar efficiency than the MTCN with the training data, but they were not able to generalize as well as the MTCN with unseen samples. On the other hand, the approximated GPR had a poor performance with the training data, but its efficiency with unssen data was very close to that of the exact GPR. As seen in Table 5.4, the disadvantage of the MTCN model is its ”long” training time. The MTCN required 3 times longer to be trained than the TCN and 23 times more than the approximated GPR. However, since the training process takes place offline, a long training time does not represent a drawback in the final implementation.

Table 5.4: Results of the GPR, TCN and MTCN architectures for the drive units Type-A.

		GPR		**TCN**	**MTCN**
		Exact	**Approximated**		
RMSE	Train	**48**	64	50	**48**
	Validation	-	-	64	**59**
	Test	76	75	71	**67**
CRA	Train	52	47	53	**54**
	Validation	-	-	46	**49**
	Test	44	44	43	**46**
Training time [s]		100	**16**	124	377

Comparison in terms of model size and computational complexity

Within this approach, not only the performance is an important factor in the selection of the regression algorithm, but also the models size and the number of MAC operations play an important role. The information about the number of parameters, the size and the amount of MAC computations of each architecture is given in Table 5.5.

As already mentioned in Section 2.5.2, the GPR is a *non-parametric* approach, which depends only on past observations to make a new prediction. The number of parameters refers to the number of observations that are stored in the model and the parameters of the covariance function. As for the TCN and MTCN models, the number of parameters refer to the weights of the convolutional and dense layers and the biases of the architectures. The model size indicates the size of the quantizied binaries of each model. The architectures were quantizied as a 16 bit floating point, which generated a negligible loss of accuracy in all models. The exact GPR model was over 10 times bigger than any other model, which made it unsuitable for many embedded systems. The MTCN was the most compact architecture by itself with a size of only 43 kBytes. Nevertheless, the deployment of deep learning models, which were trained and compiled with Tensorflow [2], requires not only the storage of the parameters in the memory unit, but also of TensorFlow Lite for microcontrollers. This framework enables the deployment of machine learning models in microcontrollers and has a size of around 16 kB [23]. The model parameters and the framework are constant values that are stored in the Flash memory. Even with the addition of the TensorFlow Lite framework, the size of the MTCN was very close to the approximated GPR, which required the least space in the flash memory.

The RAM neccesary to allocate the partial results depends on how the model performs a prediction given an input. As for the GPR, an input is multiplied by each sample within the dataset of the GPR. Nevertheless, each multiplication is added to the total result, and thus, the RAM required for the GPR models is only around 1 kB. Furthermore, the GPR models required the least amount of MAC operations to perform a new prediction, which demonstrates a lower computational complexity of these models. On the other hand, the TCN and MTCN architectures require several hundred of thousands of MAC operations to make a new prediction. However, an advantage of CNNs is their ability for parallelization. Therefore, not all the intermediate results must be saved simultaneously. Following the method to estimate the space in RAM for the TCN models, which is explained in Section 4.9.1, the MTCN requires less space in the RAM memory to perform a new prediction than the TCN.

Table 5.5: Models size, MAC operations and target platform settings [59].

	GPR		**TCN**	**MTCN**	ST Nucleo F411RE
	Exact	**Approx.**			
Number of parameters	194553	14003	10323	**8345**	
Model size [kB]	761	55	51	**42**	
Space in Flash memory [kB]	761	**55**	67	58	512
Space in RAM [kB]	**1**	**1**	54	31	128
MAC operations	134772	**12000**	380885	749620	<6MOps

In conclusion, the approximated GPR is the most compact model that can perform a new prediction with the least amount of computations. The main drawback of this model is its reduced accuracy. On the other hand, the proposed MTCN architecture delivers the most accurate results and require less space both in Flash memory and in RAM than the TCN. However, this model requires, for this specific application, more MAC operations than the other three evaluated models. All the presented models fit into the target plattform, which settings can be seen In Table 5.5.

5.2.6 Flexibility of the proposed approach for failure forecasting

So far, the results presented in this case study were obtained with the data of the Type-A DUs. To demonstrate the flexibility of this prognostic method for the implementation with diverse electric drives, the approach was applied to forecast the failures of the Type-B DUs. Due to the similarity in the data of both DU types, the same data cleansing procedure, which is explained in Section 5.2.1, was performed. The feature extraction from the raw signals and the preprocessing of these variables took place as described in Section 5.2.2. Through the feature selection stage, the following relevant features were found: *std-MTP, std-nMotor, std-Tin, peak-MTP* and *peak-nMotor*. These features were used to compute the HI and to train the GPR models. After the data labeling stage was performed, the DUs were classified as follows: 24 DUs are categorized as A group, 17 as B group and 23 as C group. The dataset with the C class DUs, which was used to train, test and validate the regression models, is shown in Table 5.6. A piecewise linear function is also used to construct the target vector. The magnitude of $maxRUL$ decreased from 300 to 250, which means that the EOL of this DU type is reached sooner once the degradation has started.

Table 5.6: Properties of the Type-B DU dataset used to train the learning algorithms.

Parameter	**GPR**	**MTCN**
Dataset	X_{red_U}, Y_U	X_{norm_U}, Y_U
Dataset size	$\{28977, 5\}, \{28977, 1\}$	$\{28991, 22\}, \{28991, 1\}$
Size train set	85[%], 19 DUs	75[%], 17 DUs
Size validation set	-	15[%], 4 DUs
Size test test	15[%], 4 DUs	5[%], 2 DUs
$maxRUL$	250	

Four models were created to estimate the RUL of the electric drives: an exact GPR, a GPR approximated with the FITC method [69], a TCN model and a MTCN architecture. In order to accelerate the search of the optimal hyperparameters of the TCN and MTCN models, the initial population of the GA contained 8 randomly generated individuals and the best 8 individuals that were obtained in the hyperparameter optimization of each model for the Type-A DUs. Then, only 20 populations were evaluated instead of 50 as with the first drive type. The optimized hyperparameters of both architectures are displayed in Table 5.7.

The predictions performed by the GPR models were smoothed with a mean moving average filter that has a window of 10. Figure 5.15 depicts the target RUL of three DUs and the predictions performed by the four evaluated models. The first chart displays the

Table 5.7: Optimized hyperparameters obtained with the GA for the Type-B DUs.

	ks	dil_{RL}	nb_{filt}	w_t	dr	η	bs	n_{dense}	$n_{flatten}$	g_N
TCN	3	2^4	9	3	0.3	1e-3	11	20	17	-
MTCN	3	2^4	7	3	0.32	1e-2	11	15	17	0.2

results of a sample that belongs to the train batch. The four models provided a precise prediction, but the TCN and MTCN architectures had a lower deviation in their outputs, although none smoothing filter was used. In the second chart, the target function of a sample corresponding to the validation set is shown. Only the predictions of the MTCN and TCN models are shown, because the GPR does not use samples for validation. Both architectures delivered a similar estimation, but the TCN had a slightly better accuracy in values close to zero. Finally, the third chart depicts the predicitions done by the four models for a sample of the test batch. The GPR models could not generalize correctly. Their estimation of the TTF was delayed. Thus, the alert for maintenance might be activated too late. On the other hand, the MTCN could predict the failure on time and its output had less fluctuations than the output of the TCN, which indicates a more reliable estimation.

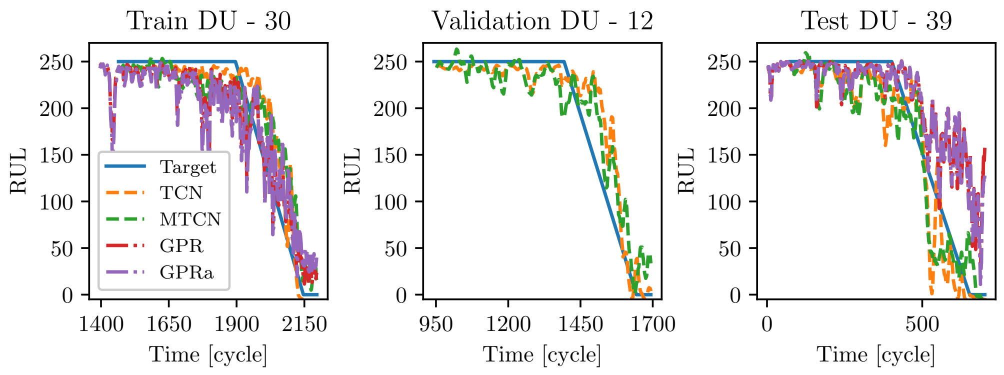


Figure 5.15: RUL predictions with TCN and MTCN models for three different Type-B DU samples.

The performance of each model as well as their size and amount of MAC operations are summarized in Table 5.8. On one side, the MTCN achieved the lowest RMSE values for the validation and test datasets as well as the highest CRA for the test dataset. This demonstrates that the MTCN generalizes better than the other evaluated models. On the other hand, the TCN had also a good performance with unseen samples and with a smaller network. Furthermore, this architecture needed around 20% less MAC computations than the MTCN. Finally, the approximated GPR resulted in a very compact model with only a 3% of the MAC computations of the MTCN. Nevertheless, this reduced model was not able to generalize as well as the MTCN and TCN architectures.

Table 5.8: Results of the GPR, TCN and MTCN architectures for the drive units Type-B.

		GPR		**TCN**	**MTCN**
		Exact	**Approximated**		
RMSE	Train	36	44	**33**	43
	Validation	-	-	51	**48**
	Test	68	67	56	**48**
CRA	Train	**50**	44	48	39
	Validation	-	-	**49**	47
	Test	44	42	41	**47**
Number of parameters		134775	12003	**5970**	7300
MAC operations		134772	**12000**	311330	385100

With the dataset of the Type-B DU it was demonstrated how flexible the developed approach is. Only with minor changes the failure prognosis of a new electric drive could be performed.

5.3 Diagnosis of faulty elements of pedelec drive units

Within this second case study, the data was collected directly in a bike during normal rides instead of on test benches. The objective is to validate the fault diagnosis algorithms with the type of data that is normal during the operation of the pedelecs. For example, here there were not only three operating settings but a wide range of possible motor performances. Thus, the algorithms were adapted for these varying settings. In this experiment, the signals that were collected were defined instead of using the variables that were available in an existing database.

5.3.1 Experimental setup description

All the experimental tests were carried out with a downhill Bike. A data logger GL1010 was connected to the drive unit to record the data for the training and validation phases of the described approach. In total, 23 CAN-signals were collected with a sampling frequency of $100Hz$. Furthermore, an external three axes accelerometer was glued to the side of the drive unit. This sensor was sampled with $24kHz$ and thus, it required an additional recording system known as SQuadriga II, which was mounted to the handlebar. This hardware was added to compare the proposed method, which uses the internal sensor signals, with a sensor type that is frequently used in condition monitoring. Figure 5.16 depicts the experimental setup. Fifteen Drive Units were tested in total, five of which were taken as reference, i.e. they did not present any symptom of a fault. The remaining drive units were prepared with two different error patterns. The first batch of five drive units had a damaged bearing; the second batch had a damaged gear. Both parts were slightly damaged so it was still possible to drive the bicycle with these drive units. A route of 25 km with slopes of over 13% was determined to perform the tests. Therefore, it was possible to cover all performance ranges of the drive unit. The route was driven twice with each sample. Contrary to failure prognosis, failure diagnosis is a task that can be accurately done with a compact ML model as a SVM.

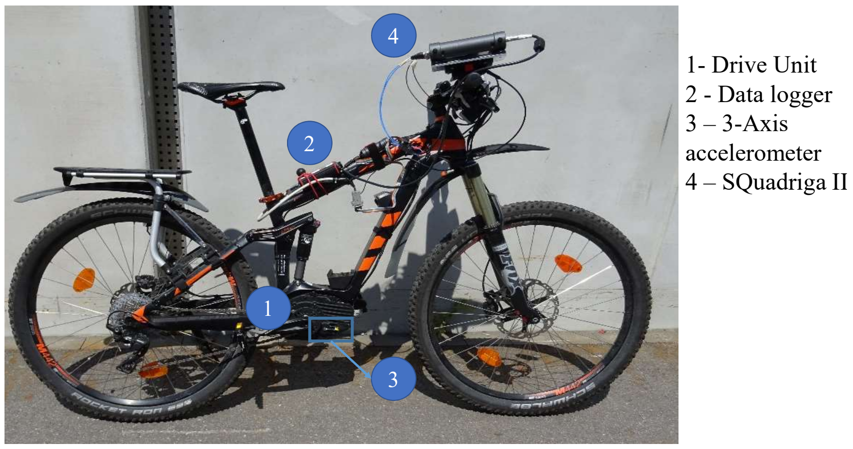


Figure 5.16: Setup used for the data generation and collection.

Data cleansing

A first filtering function removed all the information that was considered as irrelevant for the present approach. This data refers to the measurement segments where the drive unit was switched off, which occurs during the standstill of the bicycle, when the user stops pedaling or when the maximum assistance speed $(25km/h)$ is exceeded.

5.3.2 Search of optimal range

Figure 5.17 depicts the distribution of the requested motor torque of all tests after the data cleansing procedure. Contrary to the endurance test, during normal rides the maximum motor loads are rarely reached. Thus, the task was to find the range of this reference signal at which the SVM model performs better. To this end, Algorithm 7 was implemented. In total, 220 different ranges were evaluated. Since there were 3 different classes, 3 SVM models were generated. For each model, the optimal

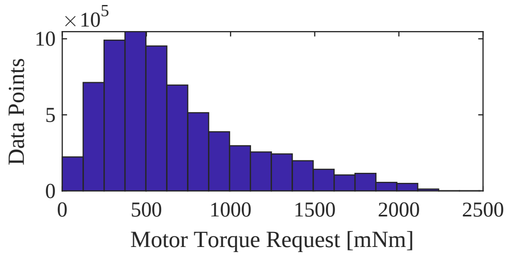
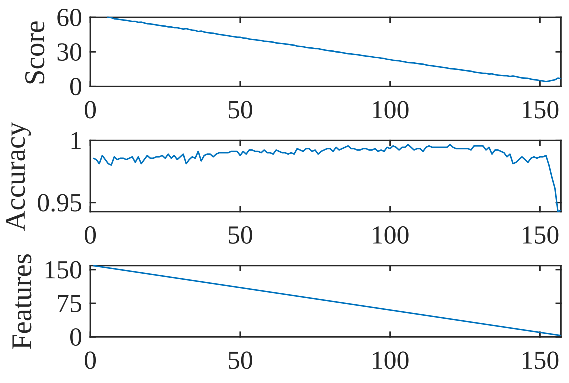


Figure 5.17: Distribution of MTR during real rides.

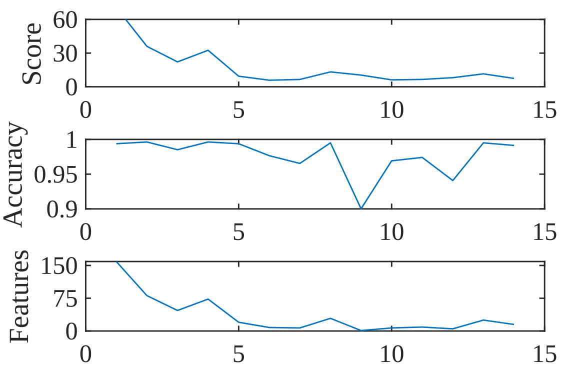
range should be located. The search of the optimal range was performed with 4 DUs of each class. The remaining samples of every class were used to test the selected range and features.

First, the following seven time-based features for each data array with a length dt of 5 seconds were computed: mean (2.1), standard deviation (2.4), peak to peak (2.11), kurtosis (2.6), skewness (2.5), RMS(2.2) and crest factor (2.12). Then, the extracted signals were normalized with the z-score normalization (2.14). After the feature generation, the number of signals increased from 23 to 161. By using the SVM-RFE, 161 SVM models had to be trained for each range, which took in average 50 seconds. With 220 ranges to be evaluated and three different classification tasks, the search with the normal SVM-RFE would take around 9 hours. On the other hand, the optimization of this method with the SA algorithm enabled the evaluation of a reduced amount of models to obtain an optimal feature subset. The search for each range was reduced to an average of 20 seconds, which gives a total of around 3.5 hours for the complete search. This is a reduction in a factor of 2.5.

Figure 5.18 displays the search with the normal SVM-RFE and with the SVM-RFE optimized by SA. This example corresponds to the search within the range $230-345mNm$ of the classes: reference and damaged gear. On one side, the normal SVM-RFE requires 160 iterations to find the best feature space. At the beginning of the search there is a



(a) SVM-RFE.

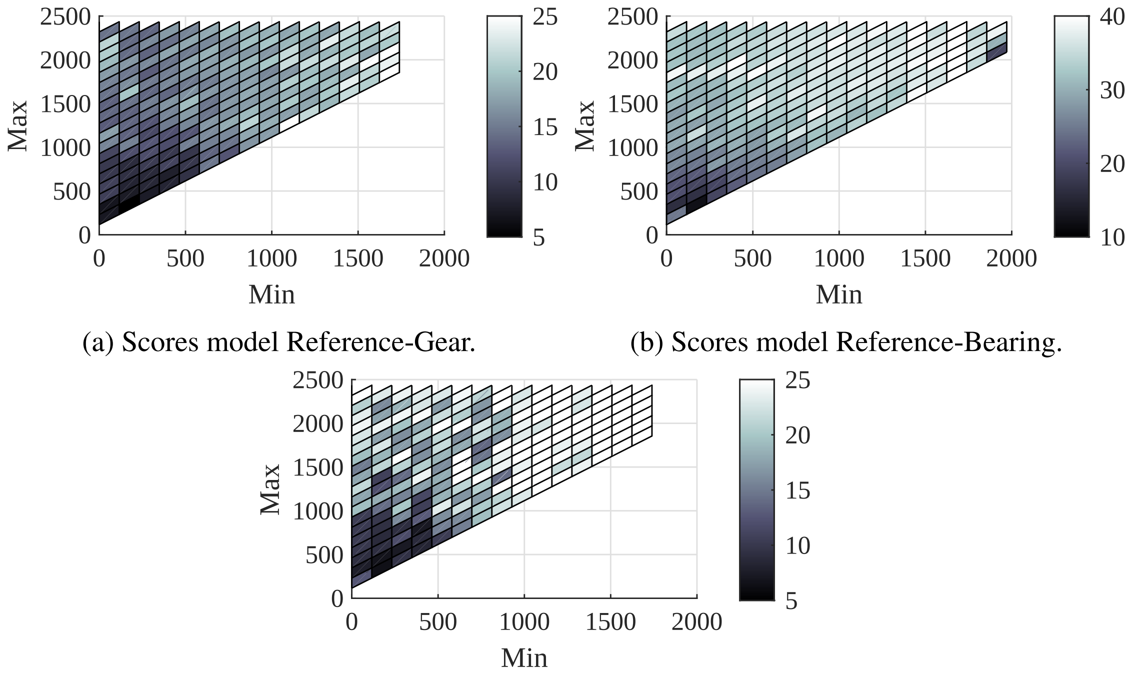


(b) SVM-RFE with SA optimization.

Figure 5.18: Dimensionality reduction within optimal range selection.

linear decrease in score. The score reduction is caused by the lower amount of features selected, and a slight increase of accuracy with less features. Only at the end of the search, when there were only 8 features or less left to train the model, the score increased due to rapid decrease in accuracy. On the other hand, this technique accelerated with the SA required only 30 iterations, from which 14 were the accepted transitions that are displayed in Figure 5.18b. The proposed dimensionality reduction technique delivered an optimal feature subset with 8 variables and a cross validation accuracy of 97.6% for the specific analyzed range.

The results of the three searches of the optimal range are depicted in Figure 5.19. The x and y-coordinates of each chart represent the minimum and maximum values of the analysed motor torque ranges, and the saturation bar to the right represents the score. Low scores represent an SVM classifier with high accuracy and a reduced number of features.



(a) Scores model Reference-Gear. (b) Scores model Reference-Bearing.

(c) Scores model Bearing-Gear.

Figure 5.19: Scores of every evaluated range for each classification task.

It can be seen that, in all cases, during the operation of the motor with low loads, the separation of the classes is more reliable than when the DU operates with high loads or when the complete torque spectrum is considered. One reason behind the large scores in the ranges with high loads is the consideration of the factor γ/N_{points} to estimate the energy E in Algorithm 7, where N_{points} is the amount of data points in the evaluated range. As seen in Figure 5.17, loads over $2Nm$ are scarce. Therefore, the data collected in this range is minimal, which increases the score. This factor assures that the selected optimal range is frequently reached by the users.

5.3.3 Results and comparison with a traditional method for fault diagnosis

In Table 5.9, the resulting optimal range and feature subset for each classification task are reported. In all cases, the range that enabled an accurate identification of the failure type comprised 115 to 230mNm. On the other hand, the optimal feature subset changed for each classification task. The features extracted from one or more motor currents (IA, IB, IC) were always present in the best feature subsets. Also the currents and voltages (id, ud, uq), which are obtained with the Park-transformation, were relevant DU signals that enabled the fault diagnosis. Finally, the motor angular speed (Nmotor) and the Y-axis of the accelerometer within the DU (AccY) provided significant information to identify certain error patterns. The motor torque together with the motor angular speed were the most relevant signals that were used to forecast the RUL of the DUs within the endurance test. Since the motor torque is derived from the currents, it is reasonable that these raw signals are also useful to diagnose the fault type.

Table 5.9: Results of the fault diagnosis approach.

Classes	**Optimal range**	**Optimal feature subset**	**Data Points**	**Score test [%]**			**Parameters**
				TPR	**TNR**	**Accuracy**	
Reference - Gear	115-230	IA(μ), IC(μ) AccY(RMS)	262	95.2	98.7	96.9	582
Reference - Bearing	115-230	Nmotor(RMS), IA(μ,κ), IC(κ) ud(μ,σ,RMS)	215	96.2	90.5	93.2	606
Bearing - Gear	115-230	IB(μ), id(η) uq(μ), AccY(μ)	267	94.3	97.1	95.2	656

The scores presented in Table 5.9 correspond to the data collected from the DUs belonging to the test batch. Each data point represents a measurement with length dt. In the results, not only the overall accuracy is considered but also the true positive rate (TPR) and the true negative rate (TNR). These metrics are important mainly if there is a data inbalance. The TPR is given by $^{TP}/_{(TP+FN)}$ and the TNR by $^{TN}/_{(TN+FP)}$, where TP are the true positives, TN the true negatives, FP the false positives and FN the false negatives. The true values refer to the correctly classified samples and the false values to the wrongly sorted samples. The worst performance was achieved by the models responsible to identify DUs with a damaged bearing. Around 10% of the measurements of the test DU with a damaged bearing were misclassified as good DUs and almost 6% were classified into the category damaged gear. The overall accuracy of the three SVM models is between 93 and 97%. To achieve a more accurate diagnosis, a fuzzy logic can be intergrated to the system, which determines the fault type only after a predefined amount of consecutive measurements have been classified to the same category.

To perform a new prediction, the SVM model requires only the samples that are support vectors and the kernel parameters. Thus, the amount of parameters that have to be stored in the microcontroller is very reduced. All the models can make efficient predictions with less than 1000 parameters. In total, around 1800 parameters are required to identify the two type of faults described in this case study. The fault diagnosis is performed with the three SVM models in a DDAG architecture, which was described in Section 4.8

Figure 5.20 depicts the correlation plot between the mean values of the motor phase currents IA and IC, which are two relevant features of the SVM model that separates DUs with damaged gears from healthy DUs. Furthermore, a SVM model was trained only with these two variables. The solid line in the chart represents the hyperplane constructed by the SVM model. Within this figure, it is seen that the selected features make a clear distintion between both classes, which simplifies the task for the selected classification model.

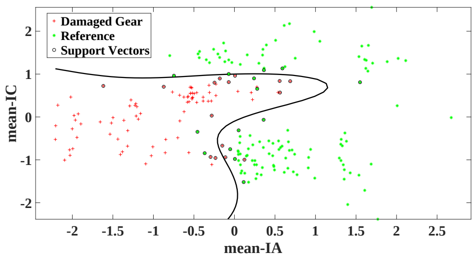


Figure 5.20: Correlation plot between two motor currents. © 2019 IEEE [101].

Finally, the fault diagnosis with the data of the external 3-Axis acceleometer was performed. To this end, the raw signals were transformed into the frequency spectrum by the FFT. Since the frequency spectrum is dependent on the rotational speed of the motor, a measurement of one minute with a driver cadence as constant as possible was done. Figure 5.21 depicts the FFT of the accelerometer Y-axis, which shows the clearest difference between the fault types. Actually, no machine learning algorithm was required to identify the error patterns with this type of sensor. As seen in this chart, the faulty samples caused high amplitudes in specific frequencies, which were different for each fault type. On the other hand, the DUs without anomalies had low amplitudes among the complete frequency spectrum. With this analysis, it is demonstrated how this type

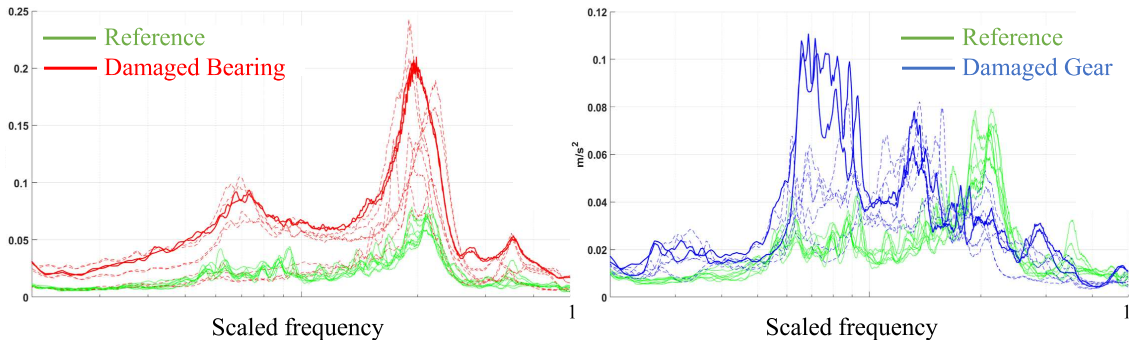


Figure 5.21: FFT of Y-Axis of the external accelerometer.

of sensors are ideal for condition monitoring of electric drives of vehicles. However, the cost of the external sensor and of the data acquisition hardware can easily reach several thousand euros. It would be unfeasible to adapt such hardware to a pedelec and even to a car. Therefore, the propossed fault diagnosis approach remains as a very efficient technique to classify the fault type of electric drives on-board.

5.4 Conclusion of the case study

Through this case study, it was possible to evaluate the effectiveness of the three core components of the developed approach, which are the data labeling, the failure prognosis and the failure diagnosis, with a real system application. First, the technique to determine the RUL of the electric drives showed consistency with the mechanical examinations that were done after the conclusion of the tests. The proposed data labeling method is an easier method to determine the magnitude of the failure than by analyzing each mechanical element. This technique is also suitable as condition monitoring method within endurance tests for example. Furthermore, the fault prognosis approach enabled an accurate estimation of the RUL of the elctric drives with machine learning models that are suitable for an implementation in the target platform . The proposed MTCN architecture demonstrated the most robust performance and this with a smaller model size than the TCN. It was also demonstrated how with minor changes, the algorithms can be adapted to forecast the failure of a different electric drive type. Finally, with the fault diagnosis approach an accurate identification of the fault type during the normal operation of the drive was achieved. This required the location of an optimal motor torque range and feature subset, which enabled the generation of a reliable model. Moreover, the approach can be a cost-effective solution to perform an on-board monitoring.

Within this work, the fault prognosis and diagnosis have been performed with two independent models. Nevertheless, the MTCN is already able to predict failures even with multiple fault types. This implies that the extracted features in the first CNN layers should contain enough information to perform a classification of the error patterns. To this end, a dense layer with a Softmax activation should be integrated to the architecture, as depicted in Figure 5.22. This new architecture would avoid the need of two independent models.

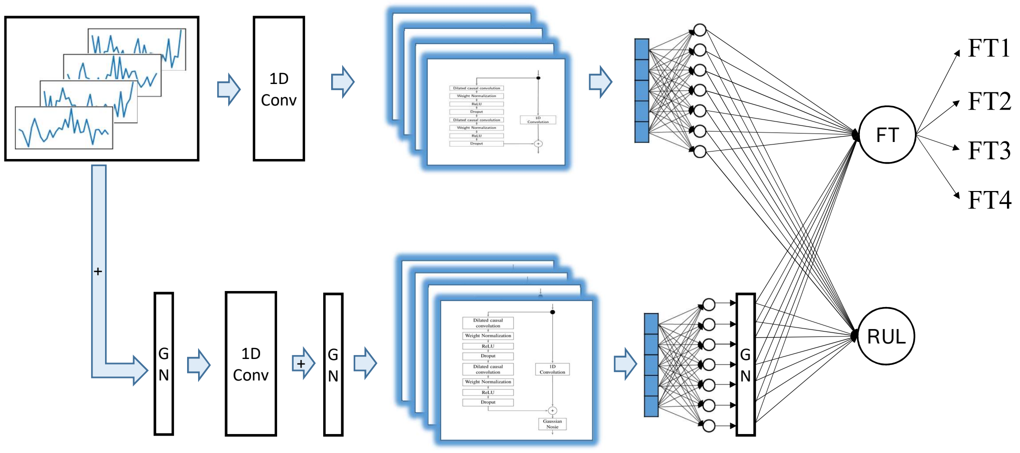


Figure 5.22: MTCN for simultaneous RUL estimation and fault diagnosis.

Chapter 6

Case study: Failure Forecasting of Aircraft Engines

The objective of this case study is to validate and compare the developed algorithms for RUL estimation against other related techniques, such as the approaches described in Section 3. To this end, the open source database known as Commercial Modular Aero Propulsion Systems Simulation dataset (C-MAPSS) [83] is used. This chapter first gives a description about the C-MAPPS database. Then both failure forecasting algorithms are implemented to predict the time to failure (TTF)) of the aircraft engines. Finally, a comparison with other machine learning approaches is given. Through this case study, it is also possible to evaluate the flexibility of the methodology to be implemented to other rotating machinery besides electric drives.

The experiments described in Section 6.2 were performed with Matlab R2019b and a 32 GB RAM. On the other hand, the experiments from Section 6.3 are performed with an NVIDIA Quadro P2000 with 5 GB, CUDA 10.2 and Tensorflow 2.1 [2]. The results of this case study have been published in [62].

6.1 Database description

C-MAPPS is a simulation framework, which is used to reproduce run-to-failure experiments of aircraft gas turbine engines. Figure 6.1 depicts the diagram of the simulated turbofan engine. Since over 1000 run-to-failure experiments are simulated, the database is vast enough to train not only traditional machine learning (ML) models but also deep learning (DL) architectures. Thus, this database has become the most common choice to train and validate learning algorithms for failure prediction.

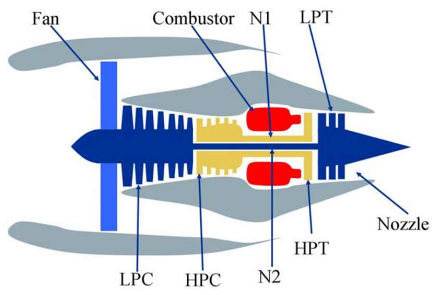


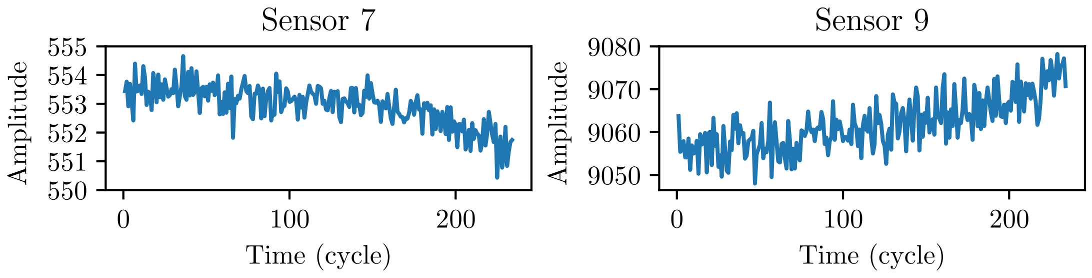
Figure 6.1: Diagram of engine simulated in C-MAPPS. © 2006 IEEE [84].

The C-MAPPS database has 4 data subsets, each of one has a different number of operating conditions, fault conditions and data length, as shown in Table 6.1. Each subset is further divided in training and test datasets.

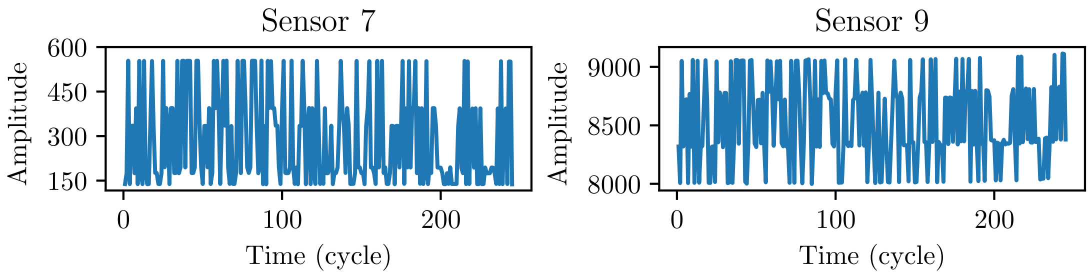
The training dataset contains the simulation of engines from an operation under normal conditions until the occurrence of the failure. On the other hand, the test data consists of segments of measurements that have taken place at an unkown time prior to the failure occurrence. The database signals include the operational settings and 21 sensor values. Figure 6.2 shows the time series of two different sensors of a sample from the subset FD1 and of a sample from the FD4 subset. From Figure 6.2a we can identify that with constant operating conditions, the time series clearly show a change on their magnitude with the time. On the other hand, the sensor values collected from turbines working under varying conditions do not have a trend that clearly indicates the change in condition with the time, as seen in Figure 6.2b.

Table 6.1: C-MAPSS Data Set

Dataset	**FD1**	**FD2**	**FD3**	**FD4**
Train trajectories	100	260	100	249
Test trajectories	100	259	100	248
Operating conditions	1	6	1	6
Fault conditions	1	1	2	2



(a) Sample from FD1 train dataset.



(b) Sample from FD4 train dataset.

Figure 6.2: Turbofan engine samples of two data subsets.

6.1.1 Performance evaluation

The main task is to predict the remaining useful life (RUL) of the test trajectories at the last available measurement. Saxena et al. [84] defined the metric Score (6.1) to compare the performance of the approaches that are developed to solve this task. This metric penalizes more heavily late predictions than early predictions. In either case, the penalty grows exponentially with increasing error. In this work, the RMSE (4.9) is also used to evaluate the performance of the proposed framework. For both metrics, the closer they are to zero the better the performance is. All the results presented for this specific case study were computed considering only the last data point of the tests trajectories, as it is indicated by the developers of the database.

$$\mathcal{S} = \begin{cases} \sum_{i=1}^{n} e^{-\frac{d_i}{13}} - 1 & \text{for } d_i < 0 \\ \sum_{i=1}^{n} e^{\frac{d_i}{10}} - 1 & \text{for } d_i \geq 0 \end{cases} \tag{6.1}$$

$$d_i = RUL_{est} - RUL_{true} \tag{6.2}$$

6.1.2 Data labeling

According to the dataset authors, the failure occured just after the last measurement of each train trajectory was performed. Thus, the data labeling procedure was reduced only to the construction of the target function. To this end the cycle time of the last measured values was selected as t_{end} in (4.6). Moreover, the creation of the piecewise target function (4.7) requires the selection of *maxRUL*. This value has a great influence in the score function. The magnitude *maxRUL* was defined emprically for each dataset.

6.2 Implementation with a GPR model

Not all stages of the methodology depicted in Figure 4.1 had to be performed in order to predict the TTF of the turbines within this database. First, the data preparation was not required, since this dataset was already structured in a similar way than the defined data structure from Table 4.1. Moreover, the time series included in this database were relatively short, with a length ranging from 20 to 550 samples. Thus, the feature extraction stage of this approach was excluded. The procedure was then reduced to the preprocessing of the sensor signals, the dimensionality reduction and the training of the GPR model.

First, the 21 sensor signals and the three operating settings were normalized with the Z-score standardization and then scaled in the range $[0, 1]$. Afterwards, the dimensionality reduction was performed according to Algorithm 1. Figure 6.3 shows the fitness values obtained for each one of the 21 sensors. Each color represents the sensor fitness for a specific dataset. As seen in the chart, the sensors had similar fitness values for the subsets

FD1 and FD3. On the other hand, the sensor measurements within the datasets FD2 and FD4 had in general lower fitness values, due to the lack of a clear trend caused by the changing operation conditions, as seen in Figure 6.2b.

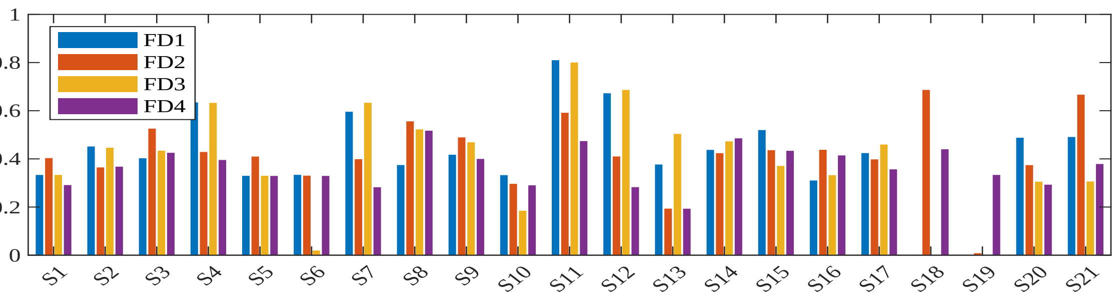


Figure 6.3: Fitness of the sensor values for each data subset.

As indicated in Algorithm 1, the feature selection depends on the factor P_s. This value specifies that the features that make P_s of the cummulative sum are chosen as the relevant features. To determine the optimal magnitude of P_s, the performance of the GPR model with ten different possible values was evaluated. To this end an exponential covariance function (Table 2.1) was selected. Figure 6.4 shows the RMSE values in dependence of P_s for the four datasets. For this example, $maxRUL$ is selected as 120 in all cases.

As seen in Figure 6.4, the RMSE remained almost constant for all datasets with P_s values between 0.7 and 1. The RMSE started its degradation from values around 0.6. The RMSE for the case $P_s = 0.7$ was almost the same as in $P_s = 1$. However, the number of features was decreased around 50%, depending on the subset. Since less features represent a lower chance of overfitting, a value of $P_s = 0.7$ was selected. In Table 6.2 the selected sensors for each dataset are shown. The data of these sensors, together with the first and second operation settings, were used as inputs to the GPR model.

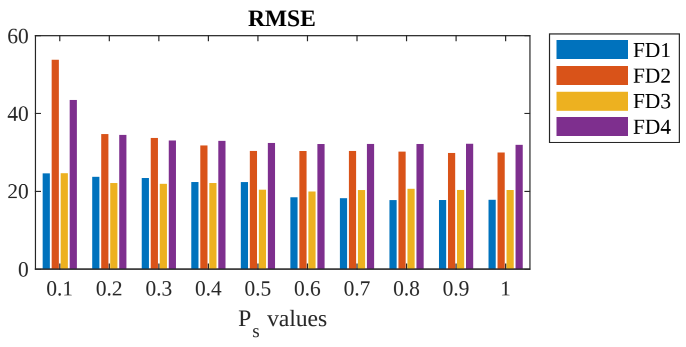


Figure 6.4: Performance change in dependence of P_s.

The value of $maxRUL$ has a great influence on the performance of the learning algorithm. In order to select an optimized quantity, values in the range $[10, 150]$ for FD1 and FD3 were evaluated. Moreover, values in the range $[50, 200]$ were assessed for FD2 and FD4. The score and RMSE values in dependence to $maxRUL$ for the four datasets are shown in Figure 6.5. The change in score and RMSE was very similar among all subsets. With high values of $maxRUL$, the model tried to predict the TTF with more cycles in advance than it was capable of. By decreasing the value of $maxRUL$, the model could more accurately predict the RUL when the TTF was equal or lower than the maximum value of the piecewise function, since this is the maximum value that the model could output. Nonetheless, the last measurement of several test trajectories took place over 150 cycles prior to the end of the useful life. Thus, the score of these trajectories increased with an exponential rate with a decreasing $maxRUL$, as observed in Figure 6.5. The best $maxRUL$ for each dataset is reported in Table 6.2. Finally, a GPR model was built for each dataset and trained using the corresponding sensors and $maxRUL$ values.

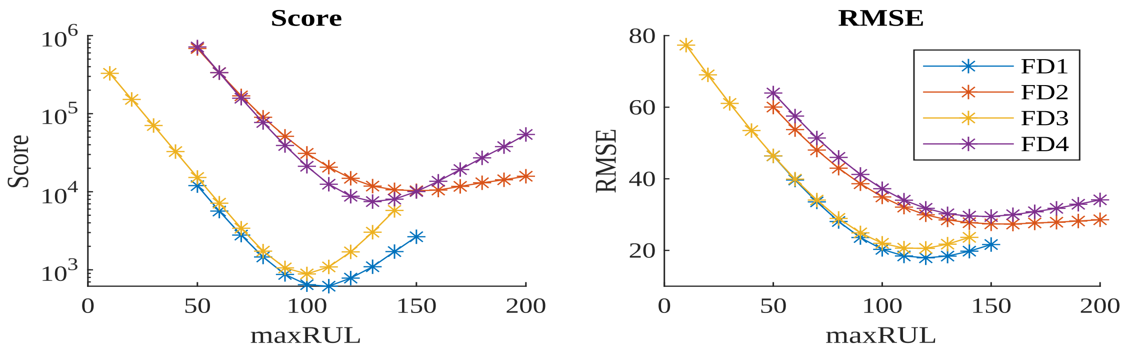


Figure 6.5: Results in dependence of $maxRUL$.

Table 6.2: Selected sensors and $maxRUL$ for each data subset

Dataset	Features	Sensors	maxRUL
FD1	11	S2, S4, S7, S9, S11, S12, S14, S15, S17, S20, S21	110
FD2	12	S3, S4, S5, S8, S9, S11, S12, S14, S15, S16, S18, S21	150
FD3	10	S2, S4, S7, S8, S9, S11, S12, S13, S14, S17	100
FD4	13	S2, S3, S4, S8, S9, S11, S14, S15, S16, S17, S18, S19, S21	130

In Figure 6.6, a predicted trajectory for each data subset is displayed. The gray area represents the 95% confidence interval that the GPR model outputs. The narrower the confidence interval, the greater the certainty that the predicted value is correct. The GPR models, which were built for the turbines operating under constant settings, delivered a more reliable RUL estimation, since the predicted value did not change considerable among two steps and the confidence interval was narrower. On the other hand, the built models for the turbines of the datasets FD2 and FD4 generated predictions with greater fluctuations and with wider confidence intervals. To reduce the effect of these variations, the output of the GPR models for FD2 and FD4 were smoothed with a one sided mean moving average filter (2.15) that had a window size of 10.

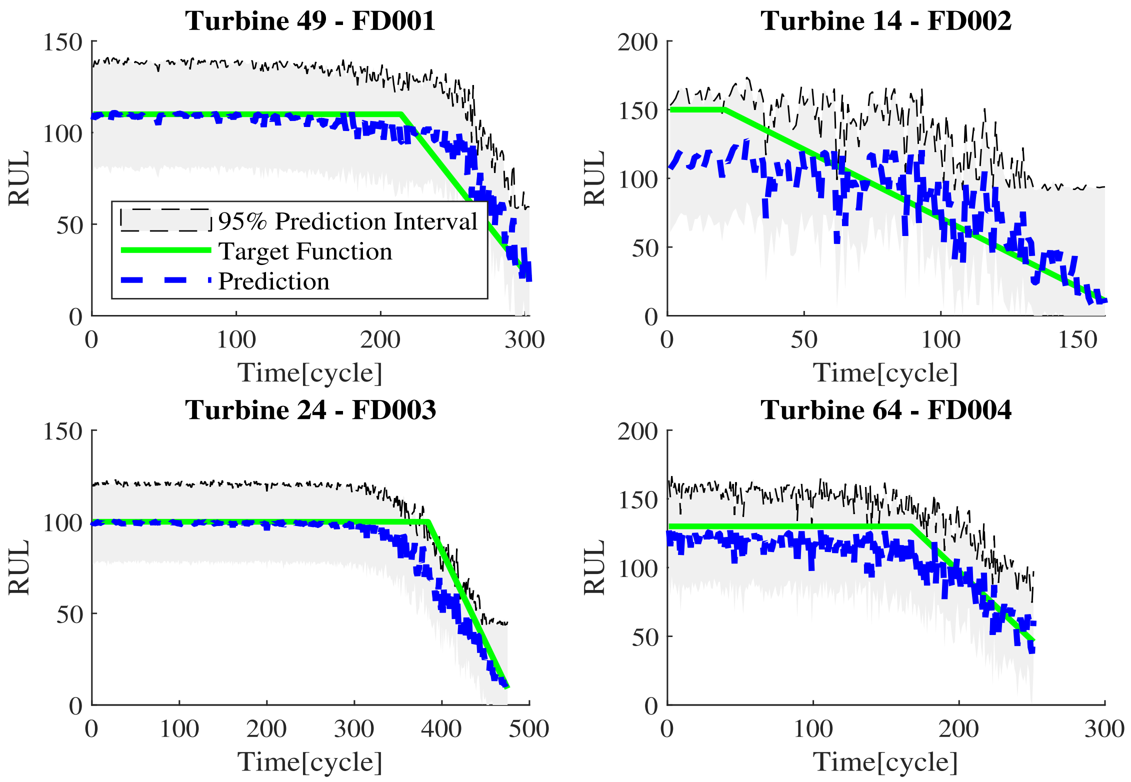


Figure 6.6: Predicted RUL of four test trajectories with the GPR approach.

In Figure 6.7, the box plot with the score values of the trajectories in the 4 subsets is shown. The median scores were between 1.2 and 4, and the maximum values of the box plots were in the range from 13 to 50. All the values above the maximum values were outliers. The outliers were mainly caused by two factors, either the target RUL was many steps above the maximum output that the model could deliver or the trajectory length was really short. Lets evaluate the specific case of FD2. Figure 6.8 depicts the

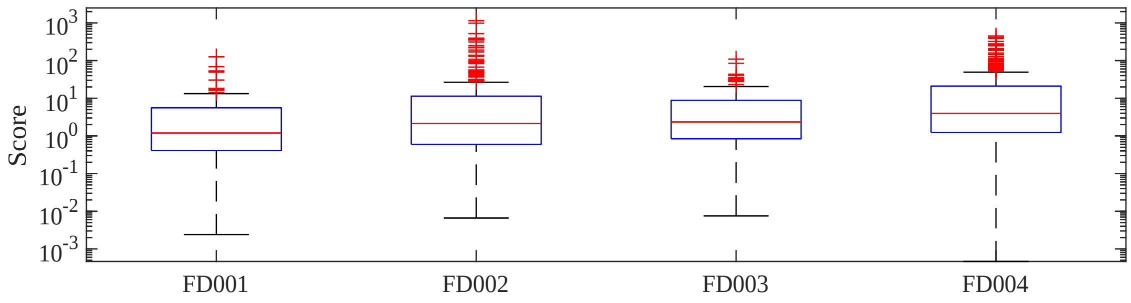


Figure 6.7: Score of every test trajectory.

score in dependence of the target RUL, the trajectory length and the difference between the predicted an real value. We can observe that indeed the trajectories, which were outliers, had a target RUL close to 200 and their length was really low. Only under 50 samples were collected for these turbines. The score of the three worst predictions was 4687. This means that 1.1% of the turbines caused 45.5% of the total score for the dataset FD2. In the last chart, it can be observed how with an increasing absolute value of the residual between predicted and target value, the score increased exponentially. From Figure 6.8, it can be concluded that the model is capable to perform accurate predictions when the end of useful life is closer. This is more relevant in a real application, than to accurately estimate the RULl hundreds or thousands of cycles in advance.

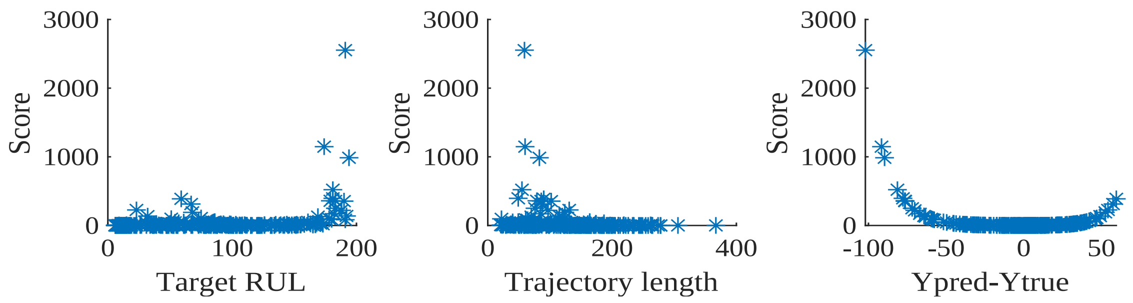


Figure 6.8: FD2 scores in dependence of target RUL and trajectory length.

In Section 6.4, the total score and RMSE values for each dataset are given. Furthermore, in that section the method is compared with other machine learning approaches proposed by other authors.

6.3 Implementation with an MTCN architecture

The presented framework for RUL estimation introduced in Section 4.7 was implemented to perform the RUL estimation of the turbines belonging to the C-MAPPS database. Only the feature extraction stage was not required due to the short length of the time series collected for each turbine. Meanwhile, the data labeling procedure for this database was explained in Section 6.1. Following the data labeling procedure explained in Section 6.1, a piecewise linear target function was used to train the deep learning model. The optimal values of max_{RUL} for each dataset found with the GPR model (Table 6.2) were as well used to label the data for the MTCN architecture.

The signals were standardized with the Z-score normalization and the features were scaled in the range $[0,1]$. The MTCN model does not require a feature selection stage and only the constant variables were removed from the datasets. Afterwards, the time series were transformed into feature maps, which were used as inputs to train the MTCN. The feature map array of the training dataset was separated into training and validation subsets. The separation was done by a ratio of aircrafts belonging to each dataset. The training dataset was composed of the measurements that belong to 85% of the engines and the validation dataset of the data of 15% of the turbines.

As seen in Figure 6.9, the minimum trajectory length of each training dataset is greater or equal that the median trajetory size of the corresponding test dataset. Thus, it was probable that during the hyperparameter optimization, a sequence length greater than the shortest test trajectory was chosen. To construct a feature map with shorter trajectories than the sequence length, a zero padding was used. Therefore, it was neccesary to perform the training of the model including feature maps with zero padding. To this end, the training and validation feature map arrays were adapted with the data regularization technique given by Algorithm 6. It was defined that the percentage of feature maps that were duplicated (p_m) was 75 for the train set and 100 for the test subset. The maximum number of rows of an image that could be filled with zeros was defined as: $dL = W - min_{Ltest}$, where W is the width of the input feature map and min_{Ltest} is the length of the shortest trajectory in the test set.

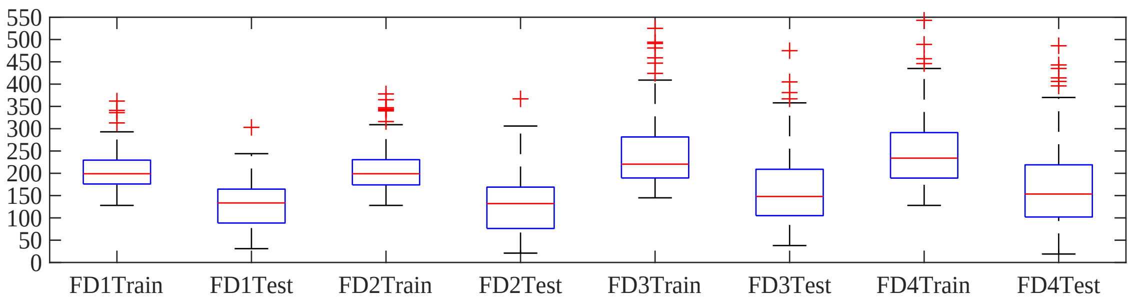


Figure 6.9: Length of the train and test trajectories of each dataset.

An MTCN model was built and trained for each data subset. Every model had the same architecture and only the training and test data changed for each learning algorithm. The base architecture was tuned by means of a genetic algorithm as detailed in Section 4.7.3. The subset FD4 was used to tune the model, since it is the most complex dataset. Moreover, to compare the performance of the MTCN against the TCN, the same procedure for hyperparameter tuning was performed to find an optimized TCN model. The first step in the HP automatic tuning is the population creation, which is performed according to Algorithm 3. This requires the definition of the minimum and maximum possible values of each hyperparameter. These limits are presented in Table 6.3. Moreover, each population P was determined to have a size of 16 individuals (n_i), each one with 10 genes n_g, which are the hyperparameters. Each individual was trained for 40 epochs and was evaluated with (4.10). Afterwards, the parents S were selected following Algorithm 4 and the crossover step took place as indicated in section 4.7.3. A determined amount of genes of the new population P_{new} were mutated following Algorithm 5. The parameters required for this algorithm were defined as follows: $mut_{rate} = 0.3$, $\Delta mut_{rate} = 0.95$ and $g_{pop} = 160$. The termination condition is given after all the desired populations have been evaluated. In this case, 30 populations were assessed for each model. Table 6.4 reports the found optimized hyperparameters for each architecture. After the optimization procedure concluded, the $maxRUL$ values were adapted to enhance

Table 6.3: Constraints of the hyperparameters for the search with the genetic algorithm.

Hyperparameter	l_{low}	l_{up}
Kernel size (ks)	2	5
Dilation rate (dil_{RL})	2^1	2^5
Number of filters (k)	2	60
Window displacement (w_t)	1	10
Dropout rate (dr)	0.05	0.75
Learning rate (lr)	1e-5	1e-2
Batch size (bs)	5	75
Number of neurons of dense layer (n_{dense})	1	200
Number of elements of the flatten layer ($n_{flatten}$)	1	20
Satandard deviation of the Gaussian noise distribution (g_N)	0	0.5

the models performance. The following limits were found as the best for both the MTCN and TCN models: *FD1-110*; *FD2-160*; *FD3-110*; and *FD4-150*.

Table 6.4: Optimized hyperparameters obtained with the GA.

	ks	dil_{RL}	k	w_t	dr	η	bs	n_{dense}	$n_{flatten}$	g_N
TCN	2	2^5	10	1	0.45	8e-3	46	162	7	-
MTCN	4	2^3	8	3	0.2	2e-3	20	20	10	0.5

Figure 6.10 depicts the estimated RUL of four different turbines performed by both the TCN and MTCN models. The displayed curves correspond to the same turbines chosen to demonstrate the suitability of GPR for RUL prediction (Fig. 6.6). As seen in Figure 6.10, both models were able to make an accurate estimation of the TTF. Moreover, the MTCN showed a better performance with RUL values under 50, i.e. this model was more accurate when the failure was closer.

In Fig. 6.11, the estimated RUL of the last measurement of each trajectory in FD4 is depicted. It can be seen that the proposed model delivered a very close prediction to the real value when the turbofan had less than 150 cycles prior to the failure occurrence. The great increase in the score was caused by the samples, from which the measurements were stopped when the RUL of the turbine was above $maxRUL$. This because we computed the scores with the real RUL of the test aircraft engines and not with the piecewise target function. Moreover, with our approach it is possible to predict the time of failure occurrence already 160 cycles in advance for FD2 and 150 cycles for FD4, which are the most complex datasets. Most approaches can predict the failure 125 cycles in advance and only in [4], authors could predict the failure occurrence with 135 cycles of anticipation for the complex datasets.

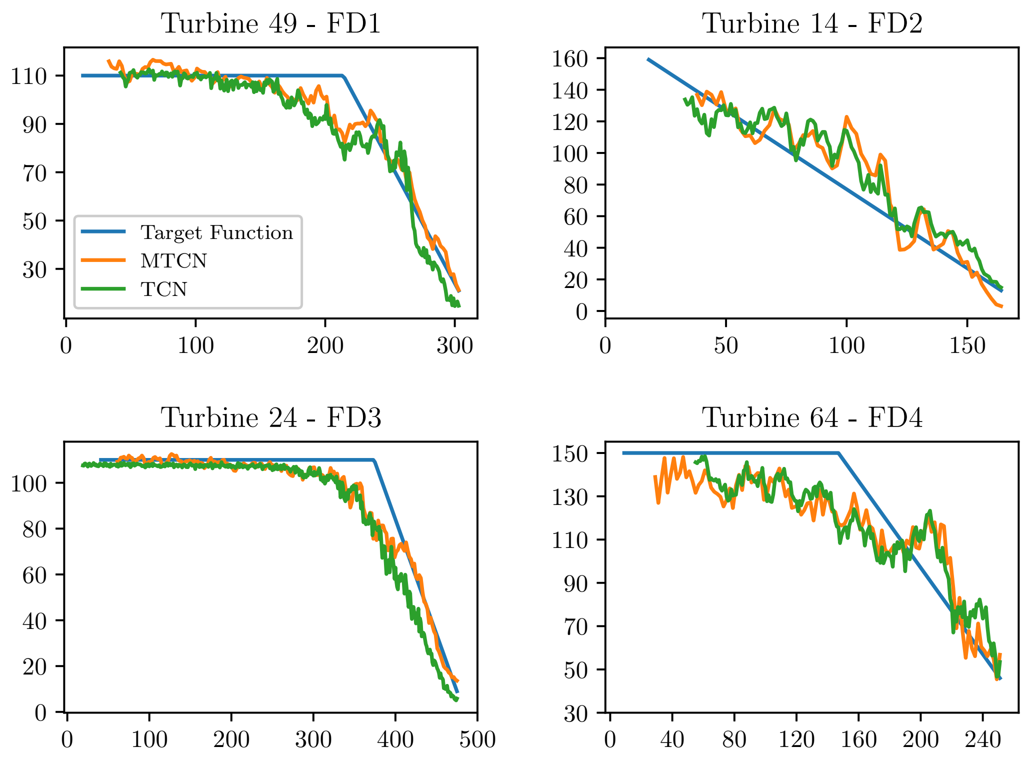


Figure 6.10: Predicted RUL of four test trajectories with the TCN and MTCN models. © 2020 IEEE [62].

The following section introduces a quantitative comparison between the TCN and MTCN models. Moreover, the performance of these architectures is compared with other deep learning models published within the last five years as well as with approaches based on traditional machine learning algorithms.

6.4 Comparison with other state of the art machine learning models for RUL estimation

The comparison of the results achieved with the proposed algorithms among other machine learning techniques for RUL estimation is shown in Table 6.5. The approaches are enlisted in order of publication year. Machine learning approaches comprise traditional models such as kernel methods or neural networks; and deep learning architectures. For the sake of simplicity, the *traditional machine learning* methods are refered as *machine learning* models within this section. The approaches enlisted in Table 6.5 are introduced in Chapter 3.

The first five methods in Table 6.5 are based on ML architectures, one of which is

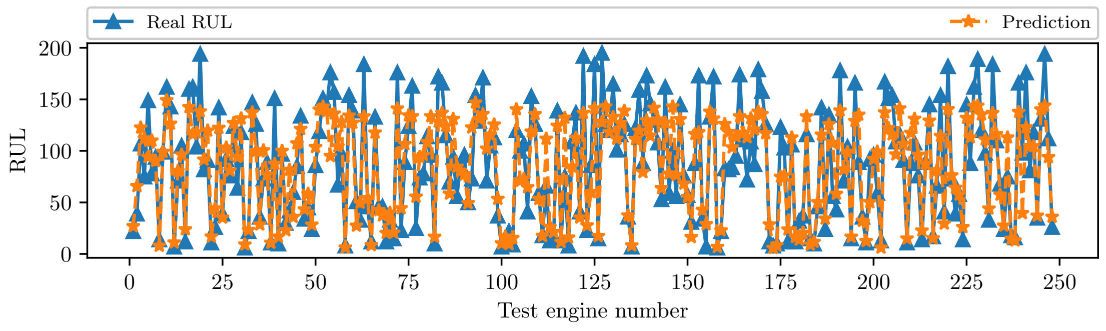


Figure 6.11: Predicted RUL by the MTCN of all test engines within the dataset FD4. © 2020 IEEE [62].

the proposed approach that uses a GPR model with an exponential kernel for the RUL estimation. Half of these methods are only tested with the first dataset, which is the least complex. The multi layer perceptron built and trained by the authors in [82], achieved the worst performance in all the datasets with scores exceeding one million for FD2 and FD4. Authors in [53], proposed a Kalman Filter ensemble to perform the RUL estimation. Though this model had an acceptable performance when a turbine operates under constant settings, but if there were multiple working conditions or multiple failure types arose, its efficiency decreased drastically. On the other hand, the developed methodology that uses a GPR model had a better score and a lower RMSE for the datasets FD2, FD3 and FD4 than the other traditional machine learning methods. Only the SVR achieved a better score in the subset FD1. Furthermore, the GPR also demonstrated a better performance when the engines operate under varying settings in comparison to the deep learning architectures based on CNNs [52, 82, 111].

The bottom eleven rows present the results of the methodologies that use deep learning models to predict the failure time of the aircraft engines. There are two main DL architectures that were implemented to solve this sequence modeling task: CNNs and LSTMs. On base of these architectures, new models with slight modifications or a combination of both methods were proposed by the authors of the published works reported in Table 6.5. It is clear how most of these approaches outperformed those using machine learning models in all subsets.

The CNN [82], and the deep CNN [52] approaches showed a great efficiency for the subsets FD1 and FD3, i.e. when the engine had only one operating setting. However, they were not able to extract relevant features from sensor signals collected during the operation under varying conditions. Thus, the CNN models had a bad performance with the datasets FD2 and FD4. Not even the inclusion of dilated convolutions [111] increased the performance of the predictive model with an operation under varying settings. Authors in [104] proposed the DSCN architecture with residual blocks that used depthwise separable convolutions instead of standard convolutions. Their architecture was able to

Table 6.5: Results for C-MAPSS data set with state of the art ML algorithms.

	Approach	**Score**				**RMSE**			
		FD1	FD2	FD3	FD4	FD1	FD2	FD3	FD4
Machine Learning	ELM [37]	1046	-	-	-	-	-	-	-
	SVR[42]	448	-	-	-	-	-	-	-
	SKF [53] ensemble	3900	5e4	2.2e5	2.8e6	25.5	30.2	36	39
	MLP [82]	17972	7.8e6	17409	5.6e6	37.5	80.0	37.4	77.3
	GPR	880	10296	993	7479	18.5	27.4	22	30.2
Deep Learning	CNN[82]	1287	13570	1596	7886	18.44	30.29	19.81	29.15
	Deep LSTM[118]	338	4450	852	5550	16.14	24.49	16.18	28.17
	BiLSTM[106]	295	4130	317	5430	13.65	23.18	13.74	24.86
	Deep CNN[52]	274	10412	284	12466	12.61	22.36	12.64	23.31
	TCN-LSTM[38]	1220	3100	1300	4000	23.57	20.45	21.17	21.03
	DSCN[104]	261	4368	247	5168	10.95	20.47	10.62	22.64
	RBM-LSTM[4]	231	3366	251	2840	12.56	22.73	12.1	22.66
	Dilated CNN[111]	-	-	-	-	12.61	28.51	12.62	30.73
	LSTM-Fusion [116]	255	1398	211	7727	11.18	16.12	10.24	21.97
	TCN	459	5540	540	4032	18,31	24,05	17,31	25,44
	MTCN	349	4728	426	3913	15.6	23.71	16.9	24.8

greatly increase the efficiency of an CNN model for the case that the engines run with changing conditions. Furthermore, the inclusion of residual blocks with dilated causal convolutions also improved the performance of CNNs for the datasets FD2 and FD4. The first experiment using a TCN architecture [38] combined the TCN with LSTM layers. This architecture was very efficient for the dataset with changing conditions but, strangely, worsened the performance of the CNN for the subsets with constant settings. Moreover, the TCN model that was built within this work achieved a performance close to the TCN-LSTM network for the FD2 and FD4 subsets, and had a much better efficiency for FD1 and FD3. Finally, the proposed MTCN architecture outperformed the traditional TCN in all data subsets. In fact, the proposed MTCN model achieved the best results in the most complex dataset among all models that used CNN layers and it was only outperformed by the LSTM model introduced in [4]. In general, the algorithms with LSTM cells [4, 106, 116, 118] demonstrated a higher efficiency than CNN models, especially for the subsets where the turbines worked with changing settings. Particularly, the semisupervised architecture with LSTMs and restricted Boltzmann machines [4] achieved the best performance in the most complex dataset. Moreover, the LSTM-Fusion provided the best result in the subsets FD2 and FD3. Nevertheless, these architectures consisted of a great amount of parameters, which as well increased the computation complexity and thus, complicates the embedded implementation of these models.

Table 6.6 indicates the number of parameters of the architectures with the best scores and their computational complexity, which is given by the number of MAC operations. The amount of parameters was obtained according to the information that the authors provided about the network architecture. Only in [104], the number of parameters was given by the authors. The MAC operations of the LSTM layers are given by (3.1). Meanwhile, the MAC operations of covolutional and depthwise separable convolutional layers were computed by equations (2.51) and (2.52) respectively. Moreover, the ratio of the best models in terms of size, computational complexity and performance for FD2 and FD4 in comparison to the proposed MTCN model are presented in Table 6.6.

Table 6.6: Comparison of the models in terms of number of parameters and MAC operations. Baseline MTCN

Approach	**Parameters**	**MAC Operations**	**Size ratio**	**MAC ratio**	**Score ratio**	
					FD2	**FD4**
BiLSTM [106]	82849	38773760	10.9	218	0.88	1.38
RBM-LSTM[4]	121264	18144264	16	102	0.72	0.73
LSTM-Fusion[116]	84201	3975500	11.1	22.3	0.3	1.98
DSCN[104]	51553	496912	6.8	2.8	0.92	1.32
TCN	14985	182382	1.9	1.03	1.17	1.03
MTCN	7553	177800	-	-	-	-

The first three models consist of LSTM layers and the last three are architectures based on CNNs. The LSTM models and the DSCN architecture achieved better results for FD2 in comparison to the proposed MTCN model. Only the RBM-LSTM [4] outperformed the MTCN model in both datasets by around 30%. However, this was achieved at the expense of a great computational complexity. The RBM-LSTM required over 18 million MAC operations, which is over 100 times the number of computations required by the MTCN. The high complexity came from the need of appending the complete measurement of a turbine in a single feature map. Since the mean size of the test trajectories within FD2 and FD4 is 150 and the number of signals is 24, the input images had an average size of $\{150,24\}$. After the restricted Bolzmann machine layer, the size was incremented to $\{150,64\}$. Thus, the computations within the first LSTM layer, which had 128 hidden units, were almost 15 million. Another factor that has to be considered for the embedded implementation of this accurate model is the great amount of parameters. Actually this model is the bigest among the 6 analyzed architectures, with 16 times more parameters than the MTCN. On the other hand, the BiLSTM[106] had an improvement of 12% in comparison to the MTCN model for FD2, but it required over 38 million MAC operations, which is over 200 times more than the operations required by the MTCN. The highest amount of computations took place at the second Bi-LSTM, due to the amount of hidden units (32) combined with the size of input feature map, which was of $\{128,150\}$. Thus, over 32 million operations were required in this layer. Moreover, the LSTM-Fusion [116] reduced the score of the MTCN by 70% for FD2.

Nevertheless, this model had a score of almost two times greater than our proposed approach in the most complex dataset (FD4). The authors that developed this architecture mentioned that the model received inputs with different window lengths, which were in the range from 10 to 100. By considering a value for W equivalent to 50, the number of MAC operations was almost 4 millions for the evaluation of a single feature map. Even though this model required only $1/10$ of computations than the Bi-LSTM, it still had a computation complexity 20 times greater than of the MTCN model.

The models with CNN layers had a reduction in the amount of MAC operations in comparison to the LSTM architectures. The DSCN [104] architecture required less than half of a million computations to achieve accurate results. In fact, it required only 3 times more MAC operations than the MTCN model and it has a slight better performance for FD2. Nevertheless, it is outperformed in the dataset FD4 by the MTCN. Moreover, our proposed architecture had almost seven times less parameters than the DSCN, which makes it more suitable for the implementation in an embedded system with a reduced memory space. Finally, the proposed MTCN also demonstrated a better performance than the TCN for both datasets. This improvement in accuracy was achieved by simultaneously halving the size of the model. However, the number of MAC operations were only reduced by 3% in comparison to the TCN.

6.5 Conclusion of the experiment

The C-MAPPS database enabled the validation of the functionality of the regression algorithms for RUL estimation. Furthermore, with this dataset it was possible to compare the efficiency of the proposed approach against other related approaches. Finally, it was demonstrated that with the proposed methodology, an accurate failure prediction can be performed even with the arise of multiple fault types and varying operation conditions.

On one hand, the results achieved by the GPR are the best among all other traditional machine learning algorithms for this database. Moreover, this model provides better results than CNNs for the complex datasets (FD2 and FD4). As already explained in Section 4.9.1, the computational complexity of the GPR is minimal, since only one multiplication takes place simultaneously and the result is added at each product. Thus, it can be easily ported into an embedded system for an online monitoring of a mechanical system. On the other hand, the proposed MTCN provides a higher accuracy than the GPR. Actually, the MTCN architecture achieves results that are similar or even better than the outcome of state of the art deep learning architectures. Moreover, the MTCN requires less parameters and less MAC operations than any other published deep learning approach for RUL estimation.

Chapter 7

Summary and Future Research

This thesis aimed to develop a comprehensive methodology that covers all the relevant stages, which are required to perform an on-board condition monitoring of electric drives. This objective posed several research questions, being the main one: How to generate accurate intelligent algorithms both for fault diagnosis as well as for failure prognosis that can be deployed in an embedded system for an on-board monitoring. After an extensive research of the state of the art, it was identified that an accurate fault diagnosis was possible with traditional machine learning algorithms. These models are normally compact and therefore easy to integrate into an embedded system. The greatest challange arose from the need to develop a model that could perform an accurate failure prognosis without having millions of parameters, which would restrict its deployment in an embedded system. Two approaches were researched to overcome this challenge. First, a Gaussian Process Regression was generated to solve the RUL estimation task. The case studies demonstrated that the GPR, together with the preprocessing steps that are part of the methodology, enabled an efficient RUL estimation even when the system operated with changing conditions. Additionally, the approach presented in this work delivered the best results for the C-MAPPS database when compared to other literature works that were based on traditional machine learning algorithms. The proposed GPR model even surpassed the performance of deep learning models like CNNs when the aircraft engine operated with varying settings, as seen in Table 6.5.

Aside from the GPR, in this thesis it was researched how a deep learning architecture based on temporal convolutions could deliver an accurate and compact model that would enable an on-board monitoring. This investigation resulted in the development of a novel deep learning architecture named Multipath Temporal Convolutional Network (Fig. 4.16). A genetic algorithm was implemented to automatically select the hyperparameters of the MTCN and its training settings, which guaranteed the obtainment of an optimized model with reduced computational time and effort. The findings of the second case study indicated that the MTCN delivered a performance that was very close to the best LSTM model but with 16 times less parameters and 100 times less MAC operations, as presented in Table 6.6. To the best of our knowledge, this network is the most compact deep learning architecture that can accurately estimate the RUL of rotating machines.

A second research question was: How to generate a vast database that could enable the validation of the proposed methodology in a real case scenario. The generation of big databases for failure prognosis is a very complex task, since electric drives can operate hundreds of hours before providing any sign of degradation. In this work, it was proposed to collect the data directly during the development phase, where electric drives go through several tests, being the endurance tests the best experimental setup to collect big data that enables the training of deep learning algorithms. The proposed methodology overcomes the challenge of data labeling for regression tasks by means of an statistical method. First, a health index that indicates the condition of each drive at every cycle is computed. The HIs of the drives are used to estimate thresholds that indicate the beginning of the degradation and the time at which the end of useful life is reached. From this point, the probability of a sudden failure is very high. Thus, the latest point in time at which the maintenance should take place is when this state is reached. The efficiency of this data labeling approach was demonstrated in the first case study as seen in Figure 5.10. Furthermore, this approach has other useful applications, such as the optimization of the time at which endurance tests are stopped, without adding any additional hardware like current condition monitoring systems.

Following the proposed data collection and labeling approaches, a propietary database was generated using sensor signals gathered in over 100 endurance tests of two types of electric drive units for pedelecs. The data belonging to the first DU type was used to prove the efficiency of the prognosis algorithms and the data labeling approach in a real case scenario, as proposed in one of the objectives of this work. The GPR delivered good estimations with a very compact model. Nevertheless, when presented with new samples, the output of the GPR presented great fluctuations,, which would hinder the decision taking based merely on its outcome. On the other hand, the MTCN proved a very high generalization capability with a compact model, which fits into the target platform as shown in Table 5.5. Furthermore, the high flexibility of the model was validated by performing the RUL estimation for the second DU type. With the GA it was possible to quickly find an optimized MTCN architecture for this new task. Tables 5.4 and 5.5 present the results of the GPR and MTCN models for the first DU type in terms of accuracy and model size. Meanwhile, the results of the prognosis models with the data of the second DU type are summarized in Table 5.8. Furthermore, a second propietary database consisting of data collected in normal rides of pedelecs with healthy and slightly damaged electric drives was used to validate the diagnosis component of the methodology. Several ranges of the motor performance were evaluated to find the operating condition at which the identification of the faults was more accurate during normal rides. The SVM models had a better performance and required less features when the electric drives operated with low torques. The findings of this case study demonstrated that also the diagnosis component of the methodology delivers reliable estimations when the monitored system operates with changing conditions.

Through these case studies it was possible to validate the suitability of the proposed comprehensive methodology in a real case scenario and compare it to other related techniques. The resulting models were accurate and compact enough, such that the decission taking regarding the time of maintenance could be based only on the predicted output, which additionally could be computed direclty in the electric drive. Furthermore, Section 4.9.2 provides a guideline to compress the size of the model and reduce the amount of MAC operations, in case that the resulting models are still too large for a determined embedded system.

The last requirement of the methodology, was that no additional hardware such as sensors or data acquisition systems should be required to perform the monitoring of the drive's conditions. The objective was to obtain algorithms, which were able to extract the relevant information directly from the existing sensors. Within the first case study, it was demonstrated that sensors that are present in electric drives such as current and rotor position sensors, as well as signals derived from them, like the motor torque and the motor angular speed, are enough to perform a reliable diagnosis and forecasting of mechanical failures. Therefore, this methodology could be easily implemented to monitor the condition of other electric drive types such as drives for electric cars.

In this work, all initial research questions were solved and all initial objectives were fulfilled. Therefore, it was possible to develop a comprehensive methodology for on-board condition monitoring that met all the requirements defined at the beginning of the thesis. In conclusion, this work proposes a method that succesfully fills the gaps in the state of the art with regard to on-board condition monitoring.

7.1 Future Research

The proposed methodology demonstrates great effiiency towards on-board condition monitoring of electric drives. However, this methodology can still be improved. Three main aspects of the methodology that can be strengthened are:

- the reduction of size and amount of MAC operations of the MTCN model;
- the reduction of the amount of data required to train the model;
- and the synergy of both condition monitoring tasks in a single model.

An interesting technique to further reduce the computational complexity and size of CNNs is by using depthwise separable convolutions [15] instead of standard convolutions. The DSCs reduce the MAC operations by a factor of ${}^{1}\!/_{C} + {}^{1}\!/_{k_c \cdot k_w}$ in comparison to standard CNNs [19]. An aspect to be considered when implementing DSCs is the loss in accuracy when the model parameters are quantized to fixed-point. As for reducing the size of the model, techniques like knowledge distillation [30] or approximate computing [63] can be researched.

As already explained throughout this thesis, the generation of big databases for failure prognosis is a complex task that could take even years. Moreover, electric drives are mass-produced components that are in continuous development. Thus, it is possible that an electric drive is replaced before enough data is collected for the generation of reliable intelligent algorithms for condition monitoring. Transfer Learning plays an important role to face this challenge, as it allows that knowledge gathered from previous system generations can be transferred to the latest developed system. In the case of a deep learning architecture, the previous knowledge refers to a model trained with the data of old system generations. By only retraining the lasts layers of the base model with data from a new, similar system, it would be possible to generate a reliable algorithm for condition monitoring with a restricted amount of data available.

The last main aspect that can be further researched is how to integrate both condition monitoring tasks into a single multi task model. In Section 5.4 an MTCN architecture that outputs both the time to failure as well as the fault type is proposed. To train this type of deep learning network, the fault type should be obtained with an statistical method, similar as with the data labeling approach for failure prognosis. Thus, mechanical inspections after each test can be completely avoided. This type of multi-task network should be investigated in more detail in upcoming works.

References

[1] Aaron, V. d. O., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio.

[2] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

[3] Abagnale, C., Cardone, M., Iodice, P., Strano, S., Terzo, M., and Vorraro, G. (2015). Power requirements and environmental impact of a pedelec. a case study based on real-life applications. *Environmental Impact Assessment Review*, **53**, 1 – 7.

[4] Andre, L. E., Emil, B., Vilmar, A., Sergey, U., and Houxiang, Z. (2019). Remaining useful life predictions for turbofan engine degradation using semi-supervised deep architecture. *Reliability Engineering and System Safety*, **183**, 240 – 251.

[5] Asghar, F., Talha, M., and Kim, S. H. (2016). Neural network based fault detection and diagnosis system for three-phase inverter in variable speed drive with induction motor. *Journal of Control Science and Engineering*, **2016**.

[6] Bai, S., Kolter, J. Z., and Koltun, V. (2018). An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *ArXiv*, **abs/1803.01271**.

[7] Bektas, O., Jones, J., Sankararaman, S., Roychoudhury, I., and Goebel, K. (2019). A neural network filtering approach for similarity-based remaining useful life estimation. *International Journal of Advanced Manufacturing Technology*, **101**, 87–103.

[8] Benkedjouh, T., Medjaher, K., Zerhouni, N., and Rechak, S. (2013). Remaining useful life estimation based on nonlinear feature reduction and support vector regression. *Engineering Applications of Artificial Intelligence*, **26**(7), 1751 – 1760.

[9] Bishop, C. M. (2006a). *Kernel methods*, chapter 6. Springer-Verlag, Berlin, Heidelberg.

[10] Bishop, C. M. (2006b). *Neural networks*, chapter 5. Springer-Verlag, Berlin, Heidelberg.

[11] Bishop, C. M. (2006c). *Probability distributions*, chapter 2. Springer-Verlag, Berlin, Heidelberg.

[12] Bishop, C. M. (2006d). *Sparse kernel machines*, chapter 7. Springer-Verlag, Berlin, Heidelberg.

[13] Cao, L., Qian, Z., Zareipour, H., Wood, D., Mollasalehi, E., Tian, S., and Pei, Y. (2018). Prediction of remaining useful life of wind turbine bearings under non-stationary operating conditions. *Energies*, **11**, 3318.

[14] Chandrashekar, G. and Sahin, F. (2014). A survey on feature selection methods. *Computers and Electrical Engineering*, **40**(1), 16 – 28. 40th-year commemorative issue.

[15] Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258.

[16] Connelly, J. A. (1993). *Low-Noise Electronic System Design*. John Wiley and Sons, Inc., USA, 1st edition.

[17] Cui, Z., Chen, W., and Chen, Y. (2016). Multi-scale convolutional neural networks for time series classification.

[18] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee.

[19] Ding, W., Huang, Z., Huang, Z., Tian, L., Wang, H., and Feng, S. (2019). Designing efficient accelerator of depthwise separable convolutional neural network on fpga. *Journal of Systems Architecture*, **97**, 278 – 286.

[20] Faber, M. (2012). *Descriptive Statistics*, chapter 3. Topics in Safety, Risk, Reliability and Quality. Springer Publishing Company.

[21] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256.

[22] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Convolutional Networks*. MIT Press. `http://www.deeplearningbook.org`.

[23] Google (2020). Tensorflowlite for microcontrollers. `https://www.tensorflow.org/lite/microcontrollers`. Accessed: 2020-08-16.

[24] Guo, S., Yang, T., Gao, W., and Zhang, C. (2018). A novel fault diagnosis method for rotating machinery based on a convolutional neural network. *Sensors*, **18**(5), 1429.

[25] Guyon, I. and Elisseeff, A. (2003). An introduction to variable and feature selection. *J. Mach. Learn. Res.*, **3**(null), 1157–1182.

[26] Guyon, I., Weston, J., Barnhill, S., and Vapnik, V. (2002). Gene selection for cancer classification using support vector machines. *Mach. Learn.*, **46**(1–3), 389–422.

[27] He, K. and Sun, J. (2015). Convolutional neural networks at constrained time cost. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE.

[28] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[29] Heimes, F. (2008). Recurrent neural networks for remaining useful life estimation. In *2008 International Conference on Prognostics and Health Management*, pages 1 – 6.

[30] Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*.

[31] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, **9**, 1735–80.

[32] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications.

[33] Huang, C., Huang, H., and Li, Y. (2019). A bidirectional lstm prognostics method under multiple operational conditions. *IEEE Transactions on Industrial Electronics*, **66**(11), 8792–8802.

[34] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift.

[35] Isermann, R. and Ballé, P. (1997). Trends in the application of model-based fault detection and diagnosis of technical processes. *Control Engineering Practice*, **5**(5), 709 – 719.

[36] "Jamie Baalis, C. (2010). *Merging Data Sources to Predict Remaining Useful Life An Automated Method to Identify Prognostic Parameters*. Ph.D. thesis, "University of Tennessee".

[37] Javed, K., Gouriveau, R., and Zerhouni, N. (2015). A new multivariate approach for prognostics based on extreme learning machine and fuzzy clustering. *IEEE Transactions on Cybernetics*, **45**(12), 2626–2639.

[38] Jayasinghe, L., Samarasinghe, T., Yuenv, C., Ni Low, J. C., and Sam Ge, S. (2019). Temporal convolutional memory networks for remaining useful life estimation of industrial machinery. *2019 IEEE International Conference on Industrial Technology (ICIT)*.

[39] Jiang, G., He, H., Yan, J., and Xie, P. (2019). Multiscale convolutional neural networks for fault diagnosis of wind turbine gearbox. *IEEE Transactions on Industrial Electronics*, **66**(4), 3196–3207.

[40] Jin, X., Sun, Y., Que, Z., Wang, Y., and Chow, T. W. S. (2016). Anomaly detection and fault prognosis for bearings. *IEEE Transactions on Instrumentation and Measurement*, **65**(9), 2046–2054.

[41] Kabzinski, J. (2017). *Advanced Control of Electrical Drives and Power Electronic Converters*, volume 75.

[42] Khelif, R., Chebel-Morello, B., Malinowski, S., Laajili, E., Fnaiech, F., and Zerhouni, N. (2017). Direct remaining useful life estimation based on support vector regression. *IEEE Transactions on Industrial Electronics*, **64**(3), 2276–2285.

[43] Konar, P. and Chattopadhyay, P. (2011). Bearing fault detection of induction motor using wavelet and support vector machines (svms). *Applied Soft Computing*, **11**(6), 4203–4211.

[44] Kramer, O. (2017). *Genetic Algorithm Essentials*. Springer Publishing Company, Incorporated, 1st edition.

[45] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

[46] Lee, J., Qiu, H., Lin, J., and Rexnord Technical Services (2007). Bearing data set.

[47] Lei, Y., He, Z., and Zi, Y. (2009). Application of an intelligent classification method to mechanical fault diagnosis. *Expert Systems with Applications*, **36**(6), 9941–9948.

[48] Lei, Y., Li, N., Guo, L., Li, N., Yan, T., and Lin, J. (2018). Machinery health prognostics: A systematic review from data acquisition to rul prediction. *Mechanical Systems and Signal Processing*, **104**, 799 – 834.

[49] Lethla, T. (2007). *Electric drives*, chapter 4. TTU, Dept. of Electrical Drives and Power Electronics, Tallinn.

[50] Li, J., Cheng, K., Wang, S., Morstatter, F., Trevino, R. P., Tang, J., and Liu, H. (2018a). Feature selection. *ACM Computing Surveys*, **50**(6), 1–45.

[51] Li, S.-Y. and Xue, L. (2018). Motor's early fault diagnosis based on support vector machine. *IOP Conference Series: Materials Science and Engineering*, **382**, 032047.

[52] Li, X., Ding, Q., and Sun, J.-Q. (2018b). Remaining useful life estimation in prognostics using deep convolution neural networks. *Reliability Engineering and System Safety*, **172**, 1 – 11.

[53] Lim, P., Goh, C. K., Tan, K. C., and Dutta, P. (2017). Multimodal degradation prognostics based on switching kalman filter ensemble. *IEEE Transactions on Neural Networks and Learning Systems*, **28**(1), 136–148.

[54] Liu, C., Zhang, L., and Wu, C. (2019). Direct remaining useful life prediction for rolling bearing using temporal convolutional networks. In *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 2965–2971.

[55] Liu, R., Yang, B., Zio, E., and Chen, X. (2018). Artificial intelligence for fault diagnosis of rotating machinery: A review. *Mechanical Systems and Signal Processing*, **108**, 33–47.

[56] MacArthur, J. and Kobel, N. (2014). Regulations of e-bikes in north america. Technical report, Transportation Research and Education Center (TREC), Portland, OR. .`https://doi.org/10.15760/trec.163`.

[57] Mahamad, A. K., Saon, S., and Hiyama, T. (2010). Predicting remaining useful life of rotating machinery based artificial neural network. *Computers and Mathematics with Applications*, **60**(4), 1078 – 1087. PCO' 2010.

[58] Maimon, O. and Rokach, L. (2010). *Data Mining and Knowledge Discovery Handbook, 2nd ed.*

[59] Mark Woods (2019). Machine learning on ultra-constrained devices. Accessed: 2023-10-08.

[60] Martin, H. and Honarvar, F. (1995). Application of statistical moments to bearing failure detection. *Applied Acoustics*, **44**(1), 67 – 77.

[61] Melendez, I., Doelling, R., and Bringmann, O. (2019). Self-supervised multi-stage estimation of remaining useful life for electric drive units. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 4402–4411.

[62] Melendez-Vazquez, I., Doelling, R., and Bringmann, O. (2020). Multipath temporal convolutional network for remaining useful life estimation. In *2020 I IEEE International Conference on Big Data (Big Data)*, pages 4137–4146.

[63] Moons, B., De Brabandere, B., Van Gool, L., and Verhelst, M. (2016). Energy-efficient convnets through approximate computing. In *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1–8.

[64] Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 807–814, Madison, WI, USA. Omnipress.

[65] Nectoux, P., Gouriveau, R., Medjaher, K., Ramasso, E., Chebel-Morello, B., Zerhouni, N., and Varnier, C. (2012). Pronostia: An experimental platform for bearings accelerated degradation tests. pages 1–8.

[66] Pan, H., He, X., Tang, S., and Meng, F. (2018). An improved bearing fault diagnosis method using one-dimensional cnn and lstm. *J. Mech. Eng*, **64**(7-8), 443–452.

[67] Patan, K. (2008). *Artificial neural networks for the modelling and fault diagnosis of technical processes*. Springer.

[68] Platt, J. C., Cristianini, N., and Shawe-Taylor, J. (2000). Large margin dags for multiclass classification. In *Advances in neural information processing systems*, pages 547–553.

[69] Quiñonero-Candela, J. and Rasmussen, C. E. (2005). A unifying view of sparse approximate gaussian process regression. *Journal of Machine Learning Research*, **6**(Dec), 1939–1959.

[70] Rafiee, J., Arvani, F., Harifi, A., and Sadeghi, M. (2007). Intelligent condition monitoring of a gearbox using artificial neural network. *Mechanical systems and signal processing*, **21**(4), 1746–1754.

[71] Rasmussen, C. E. and Williams, C. K. I. (2005a). *Approximations for GPR with Fixed Hyperparameters*. The MIT Press.

[72] Rasmussen, C. E. and Williams, C. K. I. (2005b). *Covariance Functions*. The MIT Press.

[73] Rasmussen, C. E. and Williams, C. K. I. (2005c). *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press.

[74] Ren, L., Sun, Y., Wang, H., and Zhang, L. (2018). Prediction of bearing remaining useful life with deep convolution neural network. *IEEE Access*, **6**, 13041–13049.

[75] Robert Bosch GmbH (????). Bosch ebike systems. `https://www.bosch-ebike.com/us/` Accessed 2020-08-05.

[76] Rumelhart, D. E. and McClelland, J. L. (1987). *Learning Internal Representations by Error Propagation*, pages 318–362.

[77] Saidi, L., Ali, J. B., Bechhoefer, E., and Benbouzid, M. (2017). Wind turbine high-speed shaft bearings health prognosis through a spectral kurtosis-derived indices and svr. *Applied Acoustics*, **120**, 1 – 8.

[78] Saimurugan, M., Ramachandran, K., Sugumaran, V., and Sakthivel, N. (2011). Multi component fault diagnosis of rotational mechanical system based on decision tree and support vector machine. *Expert Systems with Applications*, **38**(4), 3819–3826.

[79] Sait, A. and Sharaf-Eldeen, Y. (2011). A review of gearbox condition monitoring based on vibration analysis techniques diagnostics and prognostics. volume 5, pages 307–324.

[80] Sakar, C. O., Serbes, G., Gunduz, A., Tunc, H. C., Nizam, H., Sakar, B. E., Tutuncu, M., Aydin, T., Isenkul, M. E., and Apaydin, H. (2019). A comparative analysis of speech signal processing algorithms for parkinson's disease classification and the use of the tunable q-factor wavelet transform. *Applied Soft Computing*, **74**, 255 – 263.

[81] Salimans, T. and Kingma, D. P. (2016). Weight normalization: A simple reparameterization to accelerate training of deep neural networks.

[82] Sateesh Babu, G., Zhao, P., and Li, X.-L. (2016). Deep convolutional neural network based regression approach for estimation of remaining useful life. In S. B. Navathe, W. Wu, S. Shekhar, X. Du, X. S. Wang, and H. Xiong, editors, *Database Systems for Advanced Applications*, pages 214–228, Cham. Springer International Publishing.

[83] Saxena, A. and Goebel, K. (2008). Turbofan engine degradation simulation data set, nasa ames prognostics data repository. NASA Ames Prognostics Data Repository, `https://ti.arc.nasa.gov/tech/dash/groups/pcoe/prognostic-data-repository/`, NASA Ames Research Center, Moffett Field, CA.

[84] Saxena, A., Goebel, K., Simon, D., and Eklund, N. (2008). Damage propagation modeling for aircraft engine run-to-failure simulation. In *2008 International Conference on Prognostics and Health Management*, pages 1–9.

[85] Saxena, A., Celaya, J., Saha, B., Saha, S., and Goebel, K. (2010). Metrics for offline evaluation of prognostic performance. *International Journal of Prognostics and health management*, **1**(1), 4–23.

[86] Schröder, D. (2017a). *Intelligente Verfahren*. Springer.

[87] Schröder, D. (2017b). *Statische Funktionsapproximation*.

[88] Shalev-Shwartz, S. and Ben-David, S. (2014). *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, USA.

[89] Shao, S., McAleer, S., Yan, R., and Baldi, P. (2019). Highly accurate machine fault diagnosis using deep transfer learning. *IEEE Transactions on Industrial Informatics*, **15**(4), 2446–2455.

[90] Shen, Z., Chen, X., Zhang, X., and He, Z. (2012). A novel intelligent gear fault diagnosis model based on emd and multi-class tsvm. *Measurement*, **45**(1), 30–40.

[91] Shorten, C. and Khoshgoftaar, T. M. (2019). A survey on image data augmentation for deep learning. *Journal of Big Data*, **6**(60).

[92] Si, X.-S., Wang, W., Hu, C.-H., and Zhou, D.-H. (2011). Remaining useful life estimation – a review on the statistical data driven approaches. *European Journal of Operational Research*, **213**(1), 1 – 14.

[93] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition.

[94] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, **15**(1), 1929–1958.

[95] Tang, J., Alelyani, S., and Liu, H. (2014). *Feature selection for classification: A review*, pages 37–64. CRC Press.

[96] Thomas, W. (2011a). *Evolutionary Algorithms*, pages 253–322. CRC Press.

[97] Thomas, W. (2011b). *Genetic Algorithms*, pages 325–356. CRC Press.

[98] Thomas, W. (2011c). *Simulated Annealing*, pages 243–252. CRC Press.

[99] Tompson, J., Goroshin, R., Jain, A., LeCun, Y., and Bregler, C. (2015). Efficient object localization using convolutional networks. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[100] Vachtsevanos, G., Lewis, F., Roemer, M., Hess, A., and Wu, B. (2007). *Signal Processing and Database Management Systems*, chapter 4, pages 95–171. John Wiley and Sons, Ltd.

[101] Vazquez, I. M., Doelling, R., and Bringmann, O. (2019). Fault diagnosis approach for pedelec drive units based on support vector machines. In *2019 International Conference on Control, Automation and Diagnosis (ICCAD)*, pages 1–6.

[102] Verma, A. K., Ajit, S., Karanki, D. R., *et al.* (2010a). *Electronic System Reliability*, pages 169–191. Springer London, London.

[103] Verma, A. K., Ajit, S., Karanki, D. R., *et al.* (2010b). *Mechanical Reliability*, pages 229–266. Springer London, London.

[104] Wang, B., Lei, Y., Li, N., and Yan, T. (2019). Deep separable convolutional network for remaining useful life prediction of machinery. *Mechanical Systems and Signal Processing*, **134**, 106330.

[105] Wang, D. (2016). K-nearest neighbors based methods for identification of different gear crack levels under different motor speeds and loads: Revisited. *Mechanical Systems and Signal Processing*, **70**, 201–208.

[106] Wang, J., Wen, G., Yang, S., and Liu, Y. (2018). Remaining useful life estimation in prognostics using deep bidirectional lstm neural network. In *2018 Prognostics and System Health Management Conference (PHM-Chongqing)*, pages 1037–1042.

[107] Wang, L., Zhang, L., and Wang, X.-z. (2015). Reliability estimation and remaining useful lifetime prediction for bearing based on proportional hazard model. *Journal of Central South University*, **22**, 4625–4633.

[108] Webb, G. I. (2010). *Overfitting*, pages 744–744. Springer US, Boston, MA.

[109] Wen, Q., Sun, L., Song, X., Gao, J., Wang, X., and Xu, H. (2020). Time series data augmentation for deep learning: A survey.

[110] Wu, Y., Yuan, M., Dong, S., Lin, L., and Liu, Y. (2018). Remaining useful life estimation of engineered systems using vanilla lstm neural networks. *Neurocomputing*, **275**, 167 – 179.

[111] Xu, X., Wu, Q., Li, X., and Huang, B. (2020). Dilated Convolution Neural Network for Remaining Useful Life Prediction. *Journal of Computing and Information Science in Engineering*, **20**(2).

[112] Yu, L., Qu, J., Gao, F., and Tian, Y. (2019). A novel hierarchical algorithm for bearing fault diagnosis based on stacked lstm. *Shock and Vibration*, **2019**.

[113] Yuan, M., Wu, Y., and Lin, L. (2016). Fault diagnosis and remaining useful life estimation of aero engine using lstm neural network. In *2016 IEEE International Conference on Aircraft Utility Systems (AUS)*, pages 135–140.

[114] Yuan, Z., Zhang, L., Duan, L., and Li, T. (2018). Intelligent fault diagnosis of rolling element bearings based on hht and cnn. In *2018 Prognostics and System Health Management Conference (PHM-Chongqing)*, pages 292–296.

[115] Zhang, W., Li, C., Peng, G., Chen, Y., and Zhang, Z. (2018). A deep convolutional neural network with new training methods for bearing fault diagnosis under noisy environment and different working load. *Mechanical Systems and Signal Processing*, **100**, 439 – 453.

[116] Zhang, Y., Hutchinson, P., Lieven, N. A. J., and Nunez-Yanez, J. (2020). Remaining useful life estimation using long short-term memory neural networks and deep fusion. *IEEE Access*, **8**, 19033–19045.

[117] Zhao, H., Sun, S., and Jin, B. (2018). Sequential fault diagnosis based on lstm neural network. *IEEE Access*, **6**, 12929–12939.

[118] Zheng, S., Ristovski, K., Farahat, A., and Gupta, C. (2017). Long short-term memory network for remaining useful life estimation. In *2017 IEEE International Conference on Prognostics and Health Management (ICPHM)*, pages 88–95.

[119] Zhou, D., Li, Z., Zhu, J., Zhang, H., and Hou, L. (2020). State of health monitoring and remaining useful life prediction of lithium-ion batteries based on temporal convolutional network. *IEEE Access*, **8**, 53307–53320.

[120] Zhu, J., Chen, N., and Peng, W. (2019). Estimation of bearing remaining useful life based on multiscale convolutional neural network. *IEEE Transactions on Industrial Electronics*, **66**(4), 3208–3216.

Abbreviations

AI	Artificial Intelligence
ANN	Artificial Neural Network
CBM	Condition Based Maintenance
CL	Convolutional Layer
CM	Condition Monitoring
C-MAPPS	Commercial Modular Aero-Propulsion System Simulation
CWT	Continuous Wavelet Transform
DL	Deep Learning
DSC	Depthwise Separable Convolutions
DU	Drive Unit
EoL	End of Life
FC	Fully Connected
FD	Fault Diagnosis
FFT	Fast Fourier Transformation
FITC	Fully Independent Training Conditional
FP	Failure Prognosis
GA	Genetic Algorithm
GPR	Gaussian Processes Regression
HI	Health Index
HP	Hyper-Parameter
KNN	K-Nearest Neighbors
LSTM	Long-Short-Term-Memory
MAC	Multiply-Accumulate
MD	Mahalanobis Distance
ML	Machine Learning
MLP	Multi Layer Perceptron
MSE	Mean Square Error
MTCN	Multipath Temporal Convolutional Network
PrM	Predictive Maintenance
RAM	Random Access Memory
RB	Residual Block
RBF	Radial Basis Function
ReLU	Rectified Linear Unit
RF	Receptive Field

RFE	Recursive Feature Elimination
RMSE	Root Mean Square Error
RNN	Recurrent Neural Network
RUL	Remaining Useful Life
SA	Simulated Annealing
SVM	Support Vector Machine
TCN	Temporal Convolutional Network
TSVM	Transductive Support Vector Machine
TTF	Time To Failure
WT	Wavelet Transform

List of Tables

List of Figures

List of Algorithms