Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering

Sina Yousefi - candidate no:10010

A Data-Driven Approach for Fault Classification of a Manufacturing Process

Master's thesis in RAMS Engineering
Supervisor: Professor Shen Yin
May 2022

NTNU
Kunnskap for en bedre verden

Sina Yousefi - candidate no:10010

A Data-Driven Approach for Fault Classification of a Manufacturing Process

Master's thesis in RAMS Engineering
Supervisor: Professor Shen Yin
May 2022

Norwegian University of Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering

NTNU
Norwegian University of
Science and Technology

Summary

Fault diagnosis is among the most crucial steps in maintenance strategies to sustain the health of machine tools. Traditionally, fault diagnosis was performed based on engineers' vast expertise and technical understanding. However, advances in Machine Learning (ML) theories have decreased the role of human specialists in machine fault diagnosis, introducing Intelligent Fault Diagnosis (IFD). IFD approaches have obtained significant attention in academic and industrial applications due to their accuracy and velocity in recognizing machines' health states automatically. This research presents a novel fault identification process that uses an Extra Tree classification algorithm to classify manufacturing process defects with a feature selection approach based on feature importance. This approach is evaluated and compared against multiple machine learning algorithms, including tree-based methods, artificial neural networks, and traditional algorithms such as the Support Vector Machines (SVM). The assessment results confirm that the proposed algorithm can achieve an accuracy of above 99% in the classification task and significantly improve training time and computational resource efficiency. The proposed algorithm also enables researchers to analyze the causality of each fault based on the influential features. Further instructions to continue this line of research are correspondingly presented to enhance the proposed approach by using novel transfer learning and generative approaches.

Keywords: *Intelligent Fault Diagnosis (IFD), Fault identification, Machines, Machine learning, Deep learning*

Preface

We would like to extend our gratitude to our supervisor, Professor Shen Yin, for his constant mentoring and guidance throughout the project. In addition, the support and constant feedback of the Ph.D. student Muhammad Gibran Alfarizi have played a major role in completing this project, and therefore, we want to thank them for being of great help with their broad knowledge of machine learning.

Trondheim, May 2022

Sina Yousefi

Table of Contents

List of Tables

List of Figures

Abbreviations

AE	Auto-Encoders
AI	Artificial intelligence
ANN	Artificial Neural Networks
BN	Bayesian Networks
BPTT	Backpropagation Through Time
BRNN	Bidirectional Recurrent Neural Networks
CART	Classification And Regression Tree
CNN	Convolutional Neural Networks
DBN	Deep Belief Network
DL	Deep Learning
DT	Decision Tree
EGB	Extreme Gradient Boosting
FCNN	Fully-Connected Neural Networks
GA	Genetic Algorithms
GAN	Generative Adversarial Networks
GRU	Gated Recurrent Units
IFD	Intelligent Fault Diagnosis
IMF	Intrinsic Mode Functions
KNN	K-Nearest Neighbors
LSTM	Long Short-Term Memory
ML	Machine Learning
MLP	Multi-Layer Perceptron
PCA	Principal Component Analysis
RF	Random Forest
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Networks

RUL	Remaining Useful Life
SVC	Support Vector Classifier
SVM	Support Vector Machines
TL	Transfer Learning

Chapter 1

1 Introduction

1.1 Background

Failure of machine parts directly influences the machine's operation and can possibly endanger people's lives and result in significant financial losses. This emphasizes the need for industrial facility maintenance. Maintenance is essential for ensuring the availability and durability of manufacturing equipment, as well as product quality [1], [2]. Consequently, it is essential to identify and analyze machine components' health accurately. Effective problem identification is essential for increasing machine safety and reliability while lowering operating and maintenance costs [3]. However, considering fault diagnosis is a real-time technique involving one or more human specialists to examine machine performance, this sort of maintenance is impracticable in today's industrial facilities [4]. Model-based diagnosis techniques have been developed to solve this challenge by detecting abnormal actions and isolating problems using a model that specifies the nominal behavior of a dynamic system. On the other hand, data-driven diagnosis algorithms work only on system measurements and require relatively minimal system expertise to detect and identify system flaws [5]. Artificial intelligence (AI) approaches have quickly risen in academic and industrial areas as a burgeoning subject and a feasible alternative for

defect diagnostics among data-driven methodologies. The potential for Intelligent Fault Diagnosis (IFD) [6] to give intuitive conclusions without requiring a high degree of professional competence has piqued interest in recent years. Rather than depending on engineers' expertise and ability, these approaches use Machine Learning (ML) theories to generate machine diagnostic awareness from acquired data adaptively. IFD plans to develop diagnosis models that automatically establish a link between obtained data and machine health. Machine learning theories and expanded architectures are required for the advancement of IFD approaches [7]. Deep learning algorithms have also been used to diagnose faults intelligently and succeeded due to the availability of more efficient paradigms and more data [8].

Traditional machine learning approaches were frequently employed in IFD research from its beginning until the 2010s. Machine learning research extends to the 1950s, and artificial intelligence has seen a rise in popularity since the 1980s [9], [10]. Classic concepts, including Artificial Neural Networks (ANN) and Support Vector Machines (SVM), were established around this time. As a result of these notions, intelligent fault diagnosis emerged [6]. In these procedures, the fault indications were extracted artificially from the acquired data. Such fault indications were artificially derived from the obtained data using these procedures. The sensitive features were then used to build diagnosis algorithms that could automatically detect equipment health [6], [8]. With the help of traditional machine learning, the diagnosis models began to create a relation between the selected variables and the health conditions of machines. This reduced the need for human intervention in fault identification and ushered in the era of artificial intelligence. The primary mechanisms of machine defect diagnosis are sensor signal collection, feature extraction and selection, and fault classification [1]. Sensor signal acquisition is the

process of collecting sensor data while the equipment is running. Time-frequency analysis has traditionally derived characteristics from original sensor data in the frequency and time domains. The recovered attributes are used to train machine learning models to create fault predictions in the final fault classification step [1], [11].

Traditional fault diagnosis procedures, however, have certain disadvantages. Traditional error diagnosis systems, for instance, rely on manually picked parameters. As a result, if these manually selected features are not appropriate for the job, fault classification performance may deteriorate significantly. Furthermore, handmade features are task-specific for diverse classification tasks, implying that characteristics useful in creating accurate predictions in one circumstance are ineffectual in another. It is challenging to develop a set of features that can accurately predict outcomes in a range of situations [1], [12]. Deep learning (DL) techniques effectively overcome the above-mentioned limits due to their extensive feature learning capabilities. Deep learning has proven to be helpful in a variety of scientific and technical domains, including natural language processing [13], computer vision [14], and speech recognition [15]. DL approaches, on the other hand, are powerful enough to address challenges that are not restricted to the field of computer science [16]. Deep architectures with multiple hidden layers may learn hierarchical representations directly from raw data. Deep neural networks are built up of several layers of connected nodes, each one improving and refining the prediction or categorization. Deep architectures can use model training to automatically develop discriminative representations that will aid them in making correct predictions in subsequent classification stages based on the training data [12], [16], [17]. Deep learning methodologies have altered numerous disciplines of study, including IFD,

for more than a decade. Although IFD was able to identify machine health without the need for human fault evaluation, feature extraction before the deep learning age still depended significantly on human labor [12], [18]. Furthermore, traditional machine learning theories do not apply to the ever-growing datasets due to restricted generalization performance, reducing diagnostic precision and effectiveness [17]. Convolutional Neural Networks (CNN) [19] and Recurrent Neural Networks (RNN) [20] are two subsets of deep architectures that have gotten a great deal of interest for processing pictures and datasets with time continuity. These designs have also been used in IFD investigations to handle signals and other forms of imaging data, as well as to capture long-term data relationships [21], [22].

Although deep learning models have been successful in implementing machine fault diagnostic tasks, they still have some disadvantages [1]. First, the number of free parameters in DL models grows as the number of hidden neurons and layers increases. Creating such large networks from scratch usually demands a substantial amount of labeled data as well as a significant amount of computation and effort. Adjusting the architecture, activation functions, dropout, learning rates, and other hyperparameters has a substantial influence on performance and is a time-consuming procedure [23]. Transfer learning (TL) is a potential technique for overcoming the difficulties of training a deep architecture from the ground up [24]. Instead of fully training a neural network with the random initialization, a deep neural network that has been trained with enough labeled data in another application is utilized and fine-tuned for the job at hand. Transfer learning's primary goal is to apply what the model has learned in one context to a different but similar problem [22], [23]. Researchers may use a variety of pre-trained models to adapt to new fields, such as ResNet [25], VGG-19 [26], and Inception [27], for image classification tasks and Word2vec [28]

for textual analysis. To address data insufficiency difficulties, several researchers in IFD have begun to produce studies employing transfer learning techniques or approaches such as Generative Adversarial Networks (GAN). These approaches should provide models that can transfer diagnosis data from one or more diagnosis tasks to other related but separate issues [29]. As a result, transfer-learning theories are expected to overcome the problem of a lack of labeled instances, allowing IFD to be employed in a broader range of engineering scenarios. Furthermore, generative models, such as GANs [30] and Bayesian networks [5], may assist in producing large datasets by collecting a small number of samples, giving the necessary data to train a robust IFD model.

These three artificial intelligence-based techniques for intelligent fault diagnostics are depicted in Figure 1. The critical phases of each strategy and their benefits and drawbacks are illustrated, resulting in the use of next-generation approaches.

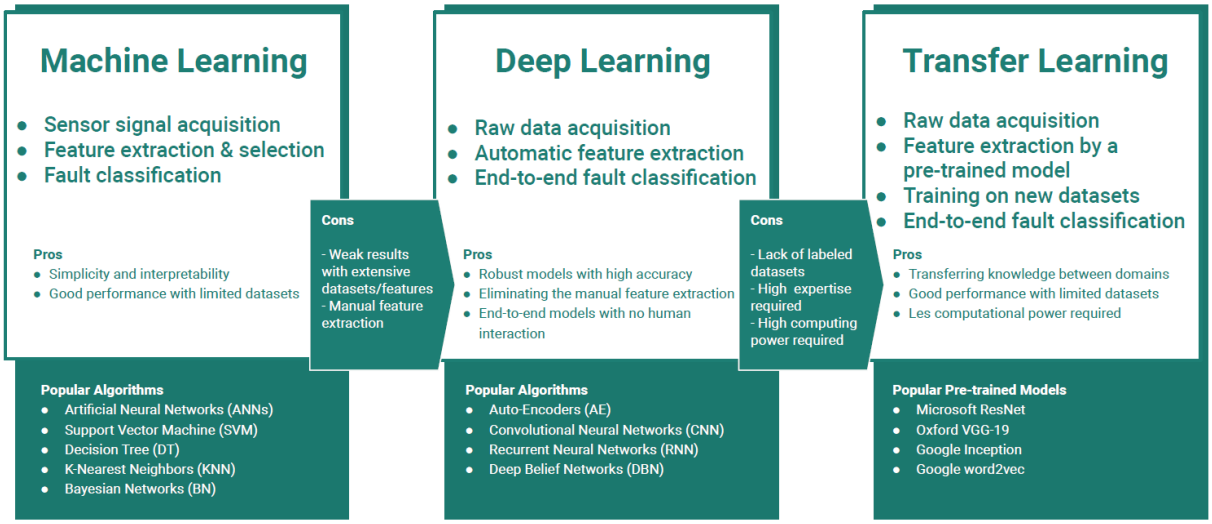


Figure 1-1. An overview of the three principal AI-based IFD approaches and their advantages and drawbacks

1.2 Objectives

The primary objective of this thesis is to propose a machine learning-based model to classify faults based on input data. This model must be robust enough to diagnose faults efficiently in the operational environment, but it is trained and tested on the PHM mechanical faults dataset for validation purposes. The proposed classification method in this thesis is based on an Extra Tree classifier, and the results are compared against several other machine learning and deep learning techniques.

This study also aims to summarize modern data-driven research works and equipment fault diagnostics' growth from theoretical and empirical perspectives. Furthermore, this thesis examines state-of-the-art approaches to overcome typical restrictions and introduces current trending topics in this domain and classic machine learning and current deep learning methods for autonomous IFD.

The following tasks are accomplished to achieve this research's primary objective:

1) Acquired the PHM dataset, preprocessed it, and divided it into three sets of training, validation, and test.
2) Built the Extra Tree machine learning classification model receiving the training dataset for completing the training process.
3) Organized a suitable evaluation process with appropriate metrics to assess the model's performance.
4) The model parameters were fine-tuned to achieve the best set of evaluation metrics.
5) The proposed method's results were compared with other approaches, such as an optimized deep learning architecture.

6) Classification results in each class were also analyzed to examine the model performance thoroughly.

1.3 Outline

The remainder of this thesis is organized in the following manner. Traditional machine learning methods and their use cases in IFD studies, the role of deep neural networks and their popular architectures in this domain, the current restrictions and limitations in performing intelligent fault diagnosis, and feasible solutions such as transfer learning methods are discussed in Section 2. Section 3 presents the proposed methodology in detail and analyzes different components such as the workflow and implementation tools. Section 4 presents and examines the experiment's findings and compares the proposed method against a variety of machine learning and deep learning models to analyze their performance on the PHM challenge dataset. Finally, section 5 sums up this experiment, recommends further stages of this study, and examines the IFD's future directions.

Chapter 2

2 Theory

2.1 Fault Diagnosis

Due to increased requirements for mechanical systems that deliver superior performance, safety, and reliability, machinery fault diagnostics are becoming increasingly important in process monitoring. Mechanical systems, such as those seen in wind turbines, airplanes, high-speed trains, and industrial machinery, have been developed due to advances in science and technology. Meanwhile, engineers must devise methods to ensure the performance of these systems, verifying that they can perform the essential functions for a specific amount of time under the stipulated conditions. Monitoring machine operating conditions, defining whether an abnormal condition or fault occurs in machines or components, determining the original cause of abnormal conditions or faults, assessing their severity, and predicting the remaining useful life or trends of abnormal conditions are just a few of these functions. One of the essential strategies for continuous maintenance is machinery fault identification, which may assist in preventing abnormal event progression, decreasing downtime, anticipating residual life, and reducing productivity loss. As a result, severe system failures and disasters can be avoided.

The critical components of mechanical equipment would inevitably generate different faults because of complex and severe conditions, including high temperature, high speed, and heavy load. Machine faults also occur and lead to severe outcomes even in sophisticated machine systems. Machinery fault diagnosis techniques involve

- observing a mechanical system over a while using occasionally sampled measures from an array of sensors,
- extracting fault-sensitive characteristics from these measurements,
- performing statistical analysis of these attributes to determine the current system health state, and
- forecasting the remaining useful life and direction of the defect.

For example, Engine Health Management (EHM) is a collection of capabilities to create customized designs that best meet the needs of individual users. An EHM system in the F135 engine provides real-time data to maintainers on the ground, drastically reducing troubleshooting and replacement time by as much as 94% over other legacy engines.

Various fault diagnostic approaches are employed in real environments to acquire meaningful data from specific physical assets. Vibration, electric current, temperature, and pressure, as well as environmental data, are all examples of machine condition monitoring data. Sensor data is pre-processed before being used for further investigation. Background noise, human influences, and sensor failures must be removed, and suitable characteristics must be computed, identified, or extracted for further fault identification. After obtaining a number of features, feature-selection methods must be used to choose the most effective characteristics

to aid the problem detection process. In this subsection, feature extraction techniques are introduced, as well as methods for diagnosing faults based on the extracted features.

2.1.1 Feature extraction approaches

Before knowledge can be gathered, data must be converted into information for proper defect identification. Fault condition indicators (features) are retrieved or chosen from the collected signals to transform waveform data into information. The common properties of reliable features are measurement at a low cost, comprehensible in physical terms, adequately definable in mathematical terms, unaffected by insignificant variables, and unrelated to other domain attributes.

Various signal processing strategies have been employed to extract important feature information and interpret signal waveform data for further problem diagnostics in motors once the spectrum data has been acquired.

2.1.1.1 Time-domain feature

Time-domain approaches are based on the waveform signal's statistically distinctive behavior throughout time. The signal's root mean square (RMS) and crest factor (CF) are the most conspicuous and essential aspects of a time-domain analysis. Variance, standard deviation, kurtosis, and skewness are the most commonly utilized attributes. These characteristics are based on signal sample distributions with time series random variables, often known as moments or instants. Because any signal change might influence the probability density function (PDF) and change the cumulate behavior, the PDF can be broken down into components in most constituent moments. As a result, monitoring this situation can yield valuable diagnostic information.

Demodulation and adaptive noise cancellation and filter-based and stochastic approaches are some alternative time-domain feature extraction techniques. One of the flaws of the time-domain feature extraction approach is the absence of observable fault symptoms, especially when a defect is still in its early stages. When extracting short-duration characteristics from a signal, this approach may be beneficial.

2.1.1.2 Frequency-domain feature

Frequency-domain characteristics can compensate for the limitations of a time-domain analysis. The knowledge that a periodic waveform signal causes a localized problem, together with characteristic frequency points and features, is used in frequency-domain approaches. Since various faults have distinct frequency domain spectrums, some changes in frequency-domain parameters may signal the existence of faults when frequency-domain characteristics are employed for fault symptom identification. Frequency-domain parameters can also detect machine defects and breakdowns early on. As a result, such indices can be employed in fault diagnosis procedures.

In frequency domains, the fast Fourier transform (FFT) is one of the most often utilized methods. A signal may be quickly transformed into the frequency domain using the FFT, a fast technique for discrete Fourier transform (DFT). If analyzing a signal in the time domain is challenging, it is much easier to convert and analyze it in the frequency domain. Several types of frequency filters, side-band structure analysis, demodulation, and descriptive representation methods are frequently employed to improve spectrum analysis results. Different forms of frequency spectra have been produced, such as power spectrum and high-order spectrum. A DFT is the

most common approach for generating a power spectrum, but other methods, such as the maximum entropy methodology, can also be utilized.

2.1.1.3 Time-frequency domain

When the signal is non-stationary, time-frequency approaches may explain machinery fault characteristics in both the time and frequency domains. The time and frequency distributions representing the signal's energy in two dimensions are used in the classic time-frequency approach. When a signal is non-stationary, the most widely utilized distribution approach is the short-time Fourier transform (STFT). The STFT is a more advanced version of the Fourier transform (FT). The target signal is divided into small windows using this method. To create succinct non-stationary signals, the width of the window function is chosen, then multiplied and shifted with the signal segment. FT is applied to each segment following the same technique to determine the signal's STFT. This graph depicts the frequency spectrum's shifting behavior as a function of time. At all relevant frequency points, STFT provides a consistent resolution.

Wavelet transform is another novel time-frequency domain approach that addresses the drawbacks of STFT. This method may also be used to analyze a signal with temporal values in a non-stationary condition. At various frequency levels, the wavelet transform gives multi-resolution.

2.1.2 Fault diagnostic methods

Several fault diagnosis approaches have been utilized for single and multiple problem diagnosis in industrial machinery systems. Signal-based, model-based, knowledge-based, and hybrid approaches are the four basic categories.

2.1.2.1 Signal-based methods

Signal processing, a branch of electrical engineering that models and analyzes data representations of physical events as well as data generated across multiple disciplines, enables modern technology that the world relies on in daily lives– including computers, radios, video devices, cell phones, and smart connected devices. As a consequence, signal processing is fundamental to the modern environment. It is where biotechnology, entertainment, and social interactions collide, and it improves the capacity to interact and exchange data. The science underpinning today's digital lives is signal processing.

For fault diagnosis, signal-based approaches rely heavily on signal processing technologies. Typically, these methods need pre-determined circumferences. Signals are influenced by their characteristics. An unexpected condition may occur when the signal or characteristics travel outside their bounds. Many signal analysis-based approaches are available, including vibration analysis, MCSA, axial flow (AF), torque analysis, noise monitoring, and impedance of inverse sequences.

Vibration levels rise when mechanical problems occur in high-speed rotating equipment. The radial forces caused by the air-gap field are the most significant sources of vibration and noise in electric devices. Vibration monitoring is a cost-effective and time-saving method of obtaining condition indicators for machine health management. The best way for defect diagnosis is vibration-based diagnostics. However, this requires costly accelerometers and accompanying wiring. This restricts its usage in various applications, particularly in tiny machines where cost is a key consideration when selecting a condition monitoring approach.

Moreover, when the diagnosis is based on numerous motors working in tandem with much noise, this constraint becomes much more complicated.

Some studies examined multi-motor fault detection approaches employing vibration analysis when motors function in isolation from the system. For feature extraction, several signal processing methods were applied. Many of these research works employed artificial neural networks to compare particular time and frequency domain characteristics. However, they never observed the diverse behavioral situations of many motors running on the same power line simultaneously.

2.1.2.2 Model-based methods

Limit or trend checking of certain observable output variables are the traditional procedures in fault identification. Model-based fault-detection approaches were created utilizing input and output signals and dynamic process models since they do not provide a deeper understanding and frequently do not allow a problem diagnosis. Parameter estimation, parity equations, and state observers are examples of these approaches. Signal model techniques have also been developed. The objective is to produce various symptoms that distinguish between nominal and defective conditions. Following fault diagnostic processes based on various symptoms, the fault is determined using classification or inference methods.

The dynamic system model is typically used in model-based fault diagnostic procedures. The actual system and model output benefit the industrial system's model-based procedures. A comparison may be performed between the simulation and actual data outputs, allowing the status of a motor to be determined through visualization. Physical modeling, system identification, and parameter estimate approaches may be utilized to create dynamic models. The most severe flaw in

model-based techniques is that the correctness of the produced model accurately represents the diagnosis system's behavior. When a system is functioning in a noisy environment, it is impossible to acquire information from the monitoring process, resulting in modeling uncertainty. In most research, model-based approaches have been utilized to gather the dynamic response of systems under normal and fault situations, but on motors separated from systems.

Model-based techniques are usually split into two sections: residual generation and decision-making. The residual results are used to guide the decision-making process. It uses independent models in both stages of fault diagnosis, which might be data-based, knowledge-based, or a combination of both analytical models. Residuals are often created using a model and pre-defined process outputs in a fault diagnostics system. However, residuals can also be generated using alternative approaches that estimate model parameter characteristics from process measurements.

2.1.2.3 Knowledge-based methods

Expert knowledge and expertise may be successfully used in knowledge-based fault detection approaches to make decisions. In other disciplines, researchers would model the link between the problem phenomena and the cause while creating the fault ontology and then apply ontology reasoning technology to diagnosis. However, there is generally an ambiguous link between the fault phenomena of the equipment to be inspected and the source of the issue during the actual fault diagnostic procedure. Knowledge-based model solutions often use a human brain-like understanding of the process for machine fault detection. The human professional specialist in real-time fault diagnostic procedures might be an engineer who applies and runs the diagnosis process and is well-versed in the strategies and techniques for

diagnosing numerous motor defects. When the signal is in a dynamic state, knowledge-based approaches rely on engineers' experience to detect the malfunction in a motor system. When signals are in complex form, these strategies can be highly beneficial in reducing the percentage of uncertainty.

Many studies based on various methodologies have been reported in the study field of defect diagnostics utilizing isolated induction motors. Due to its strong pattern recognition capacity and ability to recognize fuzzy and indefinite inputs, the artificial neural network (ANN) is possibly the most widely utilized artificial intelligence approach in motor status monitoring and problem diagnostics. The following qualities of ANN make it suitable for a wide range of applications in information fusion and problem diagnostics: neural networks have the potential to learn new things in the same manner that humans do. The learning process is carried out by altering the weight values among the neurons regularly. A neural network can also be a system with several inputs and outputs. This structure depicts how neural networks can deal with complex multi-object situations, such as numerous machine defects. The input is processed in parallel by the neural network, similar to how humans handle complex information. This unique property suggests that neural networks may spontaneously merge data from several sources simultaneously. A collection of weights is used to store the information in a trained neural network in a distributed manner. This is similar to how information is preserved in human memory. Furthermore, a neural network has a high level of fault tolerance. Its parallel structure and distributed information storage mechanism are primarily responsible for this characteristic. Therefore, the literature describes ANN as a knowledge-based approach for diagnosing single and multiple motor faults.

Diagnoses are made in these investigations by mapping different fault symptoms in a single motor to arrive at a diagnosis choice.

2.1.2.4 Hybrid methods

Since each technique of fault identification has its own set of limitations, combining numerous ways may be a helpful strategy. Several authors have proposed combining techniques like neuro-fuzzy, neural network and Bayesian interface, and DS theory with an expert system. A hybrid system termed generic integrated intelligent system architecture was suggested for equipment monitoring, problem detection, and maintenance. Different AI approaches, such as fuzzy logic and neural networks, were included in the system.

Hybrid techniques that use neural networks to assess an engine's internal health and generic algorithms to identify and quantify sensor bias can also be developed. By mixing generic methods inside the application, such a technique can employ neural networks' non-linear approximation capacity and improve the system's robustness in assessing uncertainty.

2.2 Artificial Intelligence

Artificial intelligence (AI) is the imitation of human understanding in robots that have been trained to think and act like humans. The phrase may also refer to any machine demonstrating human-like characteristics like learning and problem-solving. The capacity of artificial intelligence to rationalize and execute actions that have the highest likelihood of reaching a particular objective is its ideal feature. Machine learning is a subset of artificial intelligence that refers to the idea that computer systems can learn from and adapt to new data without human intervention.

Deep learning techniques allow for this autonomous learning by absorbing large volumes of unstructured data, including text, photos, and video. Artificial intelligence has a variety of uses. The technology may be used in multiple businesses and areas. In the healthcare business, AI is being studied and used for administering pharmaceuticals and various treatments in patients, as well as surgical operations in the operating room.

2.2.1 Machine Learning

Among all AI applications, Machine Learning (ML) is the key component. The premise of machine learning is that a computer program can learn and adapt to new data without the need for human involvement. Machine learning is a branch of artificial intelligence that maintains a computer's built-in algorithms up to date despite global economic fluctuations. Various business areas are dealing with massive volumes of data in various forms gathered from various sources. Because of the advancement of technology, particularly increased processing capabilities and cloud storage, vast amounts of data, known as big data, are becoming more readily available and accessible. Companies and governments recognize the enormous insights obtained from analyzing big data but often lack the resources and time to go through its vast amounts of data. As a result, several businesses employ artificial intelligence technologies to acquire, analyze, communicate, and exchange important information from data sets. Machine learning is a type of AI that is increasingly being used for big data processing.

Computers can solve complex scientific equations and mathematical problems in milliseconds, which is a fraction of the time it takes us to solve the same problems. They have, however, shown a lack of precision when completing our daily

behaviors, which are done naturally and spontaneously. As a result, many scientific publications are devoted to various ways of allowing computers to learn. Learning is how we ask about our environment, and robots may do the same. The machine learning method is divided into three sections: first, the algorithm calculates a pattern to model the data based on the inputs and output values supplied to the model. Following that, an error function is used to evaluate the model's performance. These metrics are used to calculate the accuracy or inaccuracy of the model's predictions. Finally, the model is tweaked to enhance accuracy while reducing error. The parameters are tweaked to minimize the discrepancy between the actual results and the model's predictions. The machine learning algorithm repeats this assessment and optimization process, which updates itself until a certain accuracy threshold is met.

We utilize codes in various programming languages to express how the machine functions in a conventional program. On the other hand, in machine learning programming, we merely create software capable of learning rules by itself to solve the given task. Computers can solve more complex problems with the information gained via this learning process. Machine Learning approaches may tackle various issues with less effort than traditional programming, which would typically need thousands of lines of code. As a result, machine learning allows us to take advantage of the computing capabilities of machines in a variety of ways.

Supervised and Unsupervised Learning are two types of machine learning algorithms [31]. Supervised machine learning algorithms may apply their learning to new data and estimate future occurrences using labeled examples. Based on a given training dataset, the learning algorithm constructs an inferred function to provide predictions about the output values. After sufficient training, the system will be able to provide objectives for each new input. The learning algorithm may also

compare its output to the correct, intended output and detect errors, allowing the model to be modified as needed. Classification and regression based on the specified outputs for each set of input features are examples of supervised machine learning tasks [24], [32]. Unsupervised machine learning approaches, on the other hand, are used when the training data is not categorized or labeled. Unsupervised learning is the study of how computers may infer a function from unlabeled data to explain hidden patterns. The system does not determine the correct output; instead, it studies the input and uses datasets to infer hidden structures from unlabeled data. Clustering, anomaly detection, and data dimensionality reduction are some of the most well-known unsupervised learning problems [24], [33].

To achieve their purpose, machine learning algorithms require training data. When the algorithm wants to evaluate its performance, it will look at the training data, classify the inputs and outputs, and re-analyze it. At the same time, the machine learning algorithm may analyze both training and validation data. Validation data is a distinct set of information [24], [31]. As long as the datasets are kept separate throughout the training and testing phase, a data scientist can cut off a piece of the training dataset for validation, also known as holdout validation. This is a test against a dataset entirely different from the one used to train the model. This sort of analysis is used to make sure the model is not underfitting or overfitting. Overfitting occurs when an algorithm can make fair judgments based on the training data but cannot properly adjust new data predictions. On the other hand, underfitting occurs when a model is not sophisticated enough to make correct predictions against both training and new data. After the validation process, data scientists can tweak hyperparameters like learning rate, input features, and hidden layers to lessen the risk of underfitting [17], [34]. A different approach for validating ML-based models

is cross-validation. Cross-validation is a resampling approach used to test machine learning models on a limited data set. The algorithm has only one parameter, k, which determines how many groups a given data sample should be split into. As a result, k-fold cross-validation is a common name for this procedure [31]. When a precise value for k is specified, it may be substituted for k in the model's reference, for example, k=10 for 10-fold cross-validation. Cross-validation is an approach utilized in machine learning to evaluate a machine learning model's performance on new datasets [35]. Moreover, there are various approaches to prevent models from overfitting the training data besides validation. For instance, dropout is a strategy for removing neurons from a neural network or ignoring them during training. In other words, distinct neurons are temporarily removed from the network. Throughout the training phase, dropout changes the notion of learning all of the network's weights to learning only a subset of the network's weights [36].

2.2.2 Feature engineering

Data handling and data preprocessing measures conducted before training models impact model performance in machine learning. Feature engineering can improve the accuracy of the same models since their data is more relevant than when all characteristics are supplied to the models. As a result, feature engineering can increase the model's overall performance. It is essential when outstanding outcomes are required for most forecasting activities. Nonetheless, mastering this process is difficult due to the fact that different types of data and datasets necessitate distinct feature engineering methodologies. The difficulties of engineering features are the primary motivation for researching algorithms that can learn features and build them automatically. While learning features may automate various jobs, feature

engineering is still one of the most effective ways to execute correctly under pressure. Feature learning methods identify the most important common patterns that distinguish classes and extract them automatically in a regression or classification procedure. Feature learning is the process of automatically engineering features using algorithms. Convolutional layers, for example, are useful in deep learning for extracting significant features from pictures and passing them to the next layer, which establishes a hierarchy of non-linear qualities that increases complexity. The final layers then use all of the characteristics that have been generated for regression or classification.

Feature selection is a subset of feature engineering, which refers to the process of independently selecting needed features. Selecting the most important independent elements that are more related to the dependent features aids in developing a dependable model. There are three types of algorithms for making feature selection: filter methods, wrapper methods, and embedding techniques.

Filter-based approaches assign each characteristic a score based on a statistical metric. The characteristics are assessed and based on their score, they are either kept or removed from the dataset. Frequently, the approaches are univariate and analyze the feature alone or concerning the dependent variable. Filter techniques include the Chi-squared test, information gain, and correlation coefficient scores. Wrapper strategies treat feature selection as a search problem in which multiple combinations are created, evaluated, and compared to one another. A predictive model evaluates a collection of features and assigns a score based on how accurate the model is. A systematic search, such as a best-first search, a stochastic search (e.g., a random hill-climbing algorithm), or heuristics (e.g., forward and backward passes) to add and delete features are all options. A wrapper technique is an example of a recursive

feature reduction procedure. While the model is being developed, embedded approaches determine which characteristics contribute the most to its validity. The most common type of embedded feature selection technique is regularization. Regularization methods, also known as penalization methods, place additional restrictions on a predictive algorithm (for example, a regression algorithm) in order to bias the model toward reduced complexity (fewer coefficients). Regularization methods include LASSO, Elastic Net, and Ridge Regression.

2.2.3 Neural Networks and Deep Learning Architectures

Artificial neural networks were created as a result of attempts to construct artificial intelligence utilizing biological neural networks (ANNs). Multiple layers of neurons linked to each other can make up a neural network. A network of linked neurons can do complex tasks, and the more neurons in the network, the more complex the activities may be performed. When they are created, all artificial neurons have specified weights and thresholds, and they link to other neurons. If a neuron's output exceeds a certain threshold, it is triggered and sends data to the next layer of the network. Deep learning techniques are defined as ANNs, which are mathematical frameworks for learning representations from data and comprise many neurons grouped in various layers. More specifically, an ANN can be considered a deep learning algorithm with more than three learnable layers.

On the other hand, a fundamental neural network is defined as a neural network with only two or three learning layers. Deep learning algorithms are taught to recognize feature hierarchies, in which lower-level characteristics combine to produce features at higher levels. In recent years, deep learning's progress has been accelerated as more vast datasets, and higher computer resources have been available. The learning

is done automatically by exposure to large training sets in modern deep learning architectures, including multiple succeeding layers.

Deep learning has transformed AI-based solutions in several domains, including computer vision and image recognition, speech recognition, natural language processing, and recommender systems research. It has also played a crucial part in a variety of industrial goods, such as virtual assistants and chatbots, and has had a significant impact on healthcare, advertising, entertainment, and a variety of other enterprises. Deep learning, for example, formulates the key components of autonomous driving and allows automobiles to learn and experiment with them in a safe setting. Many E-commerce websites, such as Amazon and eBay, utilize deep learning-based models to make recommendations that accurately forecast customers' wants based on prior visits and recommend movies, TV series, and music on entertainment platforms, including Netflix and Spotify. These models can impact a variety of topics in healthcare, including medical imaging and genomic analysis utilizing GPU-accelerated computation. Algorithms can learn the relationships between words, map them into a different language, and build machine translation systems properly. Modern language models may also generate text to describe settings, summarize articles, and converse with humans. These are only a few of the uses of cutting-edge deep learning architectures, and their importance in our daily lives is growing rapidly as more complicated models become available.

The remainder of this subsection defines the terminologies and concepts employed in neural networks and deep learning. In subsequent chapters, some more generic principles utilized in all machine learning algorithms will also be described for reference.

2.2.3.1 Neurons

Neurons are mathematical functions in an artificial neural network (ANN) that reflect functionality comparable to a biological neuron. A neuron gets several inputs, calculates the weighted average, and then sends this amount via a nonlinear function, commonly referred to as an activation function. The output of a neuron can be used as input by other neurons in another layer. The same procedure of computing the inputs' weighted sum and transformation via activation function might be repeated in other neurons. It is worth noting that these calculations are based on matrices and involve multiplying a vector of input states by a weight matrix. When a neuron receives two inputs, each input is assigned a weight. These weights are generated randomly and modified during the model's training phase. As a result, the primary purpose of model training is to find the optimal weights for the network's neurons. After the training is finished, the neural network gives greater weight to more significant inputs than those deemed less necessary. When the weight of a neuron is set to zero, that neuron's particular feature is negligible and does not influence the final output.

A linear component, referred to as the bias, is added to the input in addition to the weights of a neuron. The bias is applied to the outcome after the weight multiplication with the input, and it is used to alter the multiplied input's range. The product is provided to the neuron's activation function as the input transformation's last linear component.

2.2.3.2 Activation functions

An artificial neural network simulates the stages used by the brain to accept external stimuli, interpret the data, and provide an output, much as the brain does. When the

tasks at hand get more complicated, multiple neurons communicate to provide more precise outputs. As previously stated, artificial neurons are distinguished by their weight, bias, and activation functions. The weights and biases in the neurons perform a linear transformation depending on their inputs, and then an activation function is applied to the findings from the previous phase. The output of the activation function then travels to the next hidden layer, where the process is repeated. Forward propagation is the name given to the process of data transfer within a neural network. At the end of each training cycle, neural networks go through a back-propagation phase.

The model's error is determined using the product from forwarding propagation in this process, comprehensively covered in the following chapters. The weights and biases of the neurons are adjusted depending on this error value. However, because the model lacks an activation function, it can only perform a linear conversion on the input data using the weights and biases. Although these changes make the neural network simpler, they also make it less powerful, disabling it from learning complex patterns from the input data. As a result, the neuron's inputs are transformed using nonlinear transformation functions. The artificial neural network's non-linearity is introduced through an activation function. Many types of activation functions with distinct mathematical equations are employed in deep learning implementations.

2.2.3.3 Neural network layers

A neural network comprises neurons that are grouped into layers, as previously stated. There are three different types of layers: Each network has an input layer that receives the neural network's initial data. One or more hidden layers follow an input layer, the intermediary levels between the input and output where all calculations

are done. Finally, there is an output layer in the layer that generates outcomes for each given set of inputs. Each neuron in a fully-connected network is linked to all nodes in the layer above it and the layer below it. The weight of a neuron may be thought of as the influence of that node on the next layer's node. It is worth noting that the neurons in the hidden layers and the output layer are the only ones with activation functions, whereas the nodes in the input layer do not.

2.2.3.4 Cost function

A cost function (also known as a loss function) determines how well an algorithm models the training dataset. The goal of the training phase is usually to reduce the quantity produced by the cost function. The loss function returns a more significant value if the predictions are completely incorrect and a smaller value if they are reasonably accurate. The cost function indicates whether the model is improving or not throughout the fine-tuning phases of the method to enhance the model. Regression cost functions, binary classification cost functions, and multi-class classification cost functions are the three types of cost functions established based on the situation at hand. The most common regression functions are Mean Error (ME), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Error (MAE). Cost functions such as hinge loss, squared hinge loss, and binary cross-entropy are used for binary classification tasks. The most frequent functions for optimizing multi-class classification issues are multi-Class Cross-Entropy Loss, Kullback Leibler Divergence Loss, and Sparse Multi-Class Cross-Entropy Loss.

The degree of inaccuracy between the actual and anticipated values is quantified using cost functions, which offer this qualification as a single number. Cost functions

may be created in a variety of methods depending on the situation, and their goal is to decrease or maximize costs. The returned result is usually referred to as loss, cost, or error if the cost function is designed to be minimized. The goal of the optimization procedure is to find the optimal model parameter values such that the cost function returns the smallest possible number in this scenario. If the cost function is maximized, the value of the cost function is referred to as a reward. The objective, in this case, is to identify parameter values for which the reward is as high as achievable.

2.2.3.5 Epochs

The number of epochs is a model hyperparameter that specifies how many iterations the learning algorithm performs throughout the training dataset. Every sample in the training dataset has the opportunity to alter the intrinsic weights and biases of the neurons throughout an epoch. There can be one or multiple batches in a period. Epochs represent the total number of loops over the whole training dataset. There is another nested loop within the loop mentioned above that iterates over each batch of examples, with each batch having the same number of samples as the batch size. The batch size refers to the number of instances supplied to the model before it is updated, while the epoch refers to the total number of runs made over the whole training dataset. The number of epochs can be adjusted to any integer number between one and infinity. Historically, this number has been considerable, typically in the hundreds or thousands. This permits the method to run indefinitely until the model's error is reduced to an acceptable level. The method can continue for as long as desired. It can be terminated by using criteria other than a pre-defined number of epochs to end it—criteria such as a lack of change in the loss function's error over

time. Allowing the model to train on the training set and stopping training when performance on the validation set begins to deteriorate is a reasonable compromise. Early stopping is a practical and extensively used approach for training neural networks.

When training an extensive neural network, the model reaches a point where it stops generalizing and begins to learn the statistical noise in the training set. Overfitting is the term used in the machine learning field to describe this occurrence. Overfitting the training data raises generalization errors and reduces the model's ability to predict new data. The goal is to train the neural network long enough to learn the mapping of inputs to output data while avoiding overfitting the training data by training it too long. After each epoch, the model is evaluated on a holdout validation set in order to implement early halting during training. The training phase is terminated if the model's performance on the validation set begins to deteriorate, for example, if the loss increases or the accuracy declines.

2.3 Intelligent Fault Diagnosis (IFD)

Manual fault diagnosis and signal processing methods were once used to help determine which sorts of equipment problems existed and where they originated. These solutions, however, rely mainly on specialized knowledge, which most maintainers lack in an engineering environment. As a result, today's industrial applications choose fault diagnostic systems that can automatically identify machine health conditions [37]. Using machine learning, intelligent fault detection is expected to achieve this aim. In the past, IFD employed traditional machine learning algorithms such as Support Vector Machines to identify machine issues. The

diagnostic approach is divided into three phases: data gathering, artificial feature extraction, and health status recognition [38]. This section begins with an introduction of machine learning-based fault detection before delving more into each of the three phases. The type of information provided and the application domain determine which machine learning method is used in fault diagnosis. The following section discusses the data collecting method for IFD purposes as well as the many types of data that may be collected.

2.3.1 Data collection

During the data collection stage, sensors are mounted on devices to collect data continually. As sensor technology has evolved, many sensors such as accelerometers, currents, vibration, temperature, acoustic emission, and built-in encoders have been used for mechanical condition monitoring [39], [40]. Many researchers have used intelligent fault detection approaches to identify problem kinds by analyzing vibration signals collected by sensors in a variety of scenarios. Machine vibration signals are raw temporal signals containing valuable and ineffective information. Standard signal processing approaches for obtaining representative features from raw data include time-domain statistical analysis and wavelet transformation [22]. The signal-to-noise ratio is typically poor because vibration data is frequently hampered by transmission route and ambient noise. Other sensors, such as infrared imaging, can be used to get around this; for example, infrared imaging can give a non-contact measuring approach. Acoustic emission data may also be used to detect early bearing and gear problems and deformation, especially while running at low speeds. Diagnostics for electric-driven machine failures rely significantly on current data. This sort of data can be acquired using

only a current transformer, and it has nothing to do with equipment functioning [41], [42]. Researchers also discovered that data from multiple sensors contain complementary information that may be used to increase diagnostic accuracy when compared to data from a single sensor [22].

2.3.2 Feature extraction

Extracting essential features from the data gathered during the data collecting phase is the next stage in constructing an intelligent fault detection system using typical machine learning models. The feature extraction stage seeks to produce representative features from the recorded signals using signal processing methods such as time-domain statistical analysis, Fourier spectral analysis, and wavelet transformation [43]. These characteristics may describe mechanical health concerns, but they may also contain irrelevant or sensitive data that influence diagnosis outcomes and computational efficiency. Consequently, sensitive features are identified using dimension reduction techniques such as principal component analysis (PCA), feature discriminant analysis, and distance evaluation approach. This stage extracts several common properties from the obtained data, such as time-domain and time–frequency-domain features [41], [44]. These features include health information that represents the health of the equipment.

Following that, a feature selection procedure reduces duplicate data and improves diagnosis outcomes. Filters, wrappers, and embedded-based techniques are common ways of detecting sensitive aspects associated with machine health states. Wrapper techniques approach feature selection as a search issue involving numerous options' creation, evaluation, and comparison. A recursive feature removal technique is an example of a wrapper technique. Filter-based methods provide a score for each

attribute based on a statistical criterion. The methodologies are frequently univariate, analyzing the feature alone or concerning the dependent variable. The Chi-squared test, information gain, and correlation coefficient scores are examples of filter approaches. Embedded techniques determine which attributes contribute the most to the model's validity while being built. Regularization is the most common type of embedded feature selection approach [31], [45].

2.3.3 ML-based intelligent fault diagnosis

A brief review of IFD strategies that use machine learning models is provided in this section. When faced with unlabeled input samples, the models are trained on labeled data to determine machine health conditions. The diagnostic models are initially trained using labeled samples to achieve this purpose. For categorizing distinct types of defects, the bulk of the techniques employed in IFD research is supervised learning algorithms. On the other hand, some studies use unsupervised methods to minimize the size of their obtained datasets and make feature selections on their own. The rest of this section is divided into sections depending on the different types of algorithms and how they are used in IFD research.

2.3.3.1 Artificial Neural Networks (ANNs)

The designs of Artificial Neural Networks (ANNs) are a branch of machine learning (ML) that are at the heart of deep learning approaches. The human brain inspires their name and construction, and they function similarly to how real neurons interact with one another. A neural network comprises an input layer, one or more hidden layers, and an output layer. Each node is linked to the others and has its weight and threshold [18]. Neural networks use training data to learn and improve their accuracy over time. Once fine-tuned for accuracy, these learning algorithms become

formidable tools in computer science and artificial intelligence, allowing us to categorize and cluster data swiftly. Voice recognition and visual recognition activities can take minutes rather than hours when compared to manual identification by human professionals [24].

Although deep learning architectures have been used in the majority of IFD research utilizing ANNs, there have been occurrences of shallow neural networks being used to identify defects based on input data. These studies are classified as classic machine learning approaches, and they are studied in this area, whereas publications that use current deep neural networks are examined in the following sections. Bernieri et al. for example, conducted research. For example, Bernieri et al.'s [46] research were one of the first to use ANNs for dynamic online problem diagnosis. They demonstrated that artificial neural networks could help with system identification and flaw detection in applications that need a fast response time. Another advantage of using neural networks, according to this article, is that they are a general approach for nonlinear dynamic system applications that would otherwise need ad-hoc solutions. ANNs were used in numerous current approaches to model systems, identify problems, predict defect prognosis, and estimate the machines' remaining usable life, according to Tung et al. [2]. (RUL). The majority of ANN strategies were single-step forward time series forecasting models, suggesting that deep neural networks were not being employed when this research was published, i.e., in 2009. Lei et al. [43] proposed a two-stage learning technique for intelligent machine failure detection. It is based on the idea of unsupervised feature learning and uses artificial neural networks to learn features from raw data. In the method's initial learning phase, sparse filtering, an unsupervised two-layer neural network, is used to extract features from mechanical vibration data

dynamically. SoftMax regression is used in the second stage to identify health conditions based on learned features. Mostefa Khelil et al. [10] developed an intelligent system that can detect and distinguish three recurrent events in a PV array, including healthy and short circuit failures and string disconnection, using artificial neural networks. The built model has a simple learning curve and only uses four inputs. On a small grid-connected PV generator (PVG), experimental validation of the proposed IFD was done, demonstrating that this approach can adequately identify and categorize existing faults with over 98 percent accuracy.

2.3.3.2 Support Vector Machine (SVM)

One of supervised machine learning models for solving classification issues are support vector machines. The goal of the support vector machine algorithm is to find a hyperplane in N-dimensional space (where N is the number of features) that separates data points. There are numerous hyperplanes from which to pick to divide the classes of data points. The aim is to find a plane with the highest significant margin or distance between data points from all classifications. [31], [47]. Hyperplanes are decision boundaries that help categorize data. Different categories can be applied to data points on each side of the hyperplanes. Maximizing the margin distance provides some reinforcement, making the following data points simpler to categorize. The margin of the classifier is improved by using these support vectors. These are the considerations that will aid in the creation of the SVM [32], [48].

The first attempts to employ SVM for machine status monitoring and problem diagnosis were made in the late 1990s. With various kernel functions and cross-validation, SVM-based fault diagnostic approaches were later recommended in several research works, demonstrating better fault detection capabilities over

standard machine learning algorithms [49]. For instance, Samanta [50] extracted features from vibration signals from a spinning machine with standard and problematic gears. The obtained features were provided as inputs to both ANN-based and SVM-based classifiers. To increase the number of nodes in the hidden layer and the input feature selection in SVMs, genetic algorithms (GA) were applied. The findings showed that in the majority of circumstances, SVM outperforms ANN in terms of classification accuracy. Sugumaran et al. [51] used a Decision Tree to pick the best qualities from a sample group for classification and the Proximal Support Vector Machine (PSVM) to categorize flaws based on statistical data in another investigation efficiently. PSVM and SVM were used to extract and categorize statistical features, and the results were compared. Zhang et al. [52] proposed using ensemble empirical mode decomposition to decompose the vibration signal into a set of intrinsic mode functions (IMFs) when a bearing has faults. The first five IMFs' permutation entropy (PE) values (IMF-PE) are computed to reveal the multi-scale intrinsic features of the motor bearing's vibration signal. Support vector machines (SVM) optimized by inter-cluster distance are then used to classify the fault type and severity (ICDSVM). Vibration analysis, acoustic approaches, and SVM-based pipeline leakage detection were all examined by Datta et al. [53]. In this study, the benefits and drawbacks of each technique are outlined, and all approaches are evaluated based on their applicability. On the other hand, acoustic reflectometry is the most effective technique since it can identify obstacles and leaks in pipes as small as 1% of their diameter. The most current research on machine learning models for wind turbine status monitoring was described by Stetco et al. [54]. The majority of models uses a dataset called SCADA or simulated data, with classification accounting for around two-thirds of the techniques and regression accounting for the

rest. This study's most often utilized techniques were neural networks, support vector machines, and decision trees. SVMs, according to the data, are among the best acceptable models in most situations, with over 90% accuracy.

2.3.3.3 Decision Tree (DT)

The supervised learning algorithms family includes the Decision Tree (DT) algorithm. Unlike other supervised learning algorithms, the decision tree technique may be used to address regression and classification problems. The goal of using a Decision Tree is to create a training model that can predict the class or value of the target variable. As the name indicates, a DT uses a tree-like flowchart to depict the predictions that arise from feature-based divides [55]. A root node initiates the process, which concludes with a leaf decision. When utilizing Decision Trees to forecast a record's class label, we start at the top of the tree. A decision-tree-based model is an effective supervised methodology for applying classification algorithms in high-dimensional data [54].

For fault identification, security evaluation, and system control in power systems, DT models have been employed in a number of studies. Because the decision tree is simple to understand and grasp, its accuracy in flaw identification may be demonstrated using both testing data and expert knowledge [56]. [57], for example, describes a defect detection and categorization system based on decision trees. As characteristics in the training and test sets, widely available data from solar photovoltaic (PV) systems are used, including PV array voltage, operating temperature, and irradiance. The trained DT models showed excellent defect detection and classification accuracy in experiments. In another study by Yan et al. [56], used the classification and regression tree (CART) approach for decision tree

induction as a data-driven diagnosis tool for AHUs. The method comprised a steady-state detector and a regression model to increase the diagnostic technique's interpretability. This approach was shown to have excellent diagnostic performance, with an average F-measure of 0.97.

In their most basic form, decision trees are algorithms that are easy to perceive and understand. On the other hand, these models may be overly simple for issues with more complicated aspects. As a result, several tree-based algorithms have been developed to improve accuracy while preserving processing efficiency [4]. Ensemble approaches, in which many decision trees are combined to obtain better prediction performance than a single decision tree, are gaining popularity. The main principle behind the ensemble model is that a group of weak learners join forces to create a strong learner. The most common methods for creating ensemble decision trees are boosting and bagging. The XGBoost and Random Forest (RF) algorithms, which are often used in IFD investigations, are explained in the following subsections as examples of such approaches.

2.3.3.3.1 Extreme Gradient Boosting (XGBoost)

The gradient boosting decision-tree-based ensemble machine learning technique XGBoost, or eXtreme Gradient Boosting, is a gradient boosting decision-tree-based ensemble machine learning approach. XGBoost refers to the technological goal of pushing boosted tree algorithms' computing resources to their limits [58]. In applications involving small-to-medium structured/tabular data, a decision tree-based method can outperform more complicated models such as ANNs. The algorithm's implementation was created to reduce computing time and memory requirements. One of the design aims was to maximize available resources

throughout the model's training phase. The two main reasons for using XGBoost models are execution speed and model performance [47].

When large datasets are unavailable, and the input characteristics are not in the form of pictures, Extreme Gradient Boosting may be a feasible option. As a result, specific IFD research articles have used this strategy while working with structured and numerical datasets. For example, Zhang et al. [59] introduced a novel technique based on signal processing using an XGB algorithm that only utilizes phase voltage and current data. There are three fault identification results to classify within this study: no-fault, zero-line fault, and ground fault. The XGBoost technique was developed by combining preprocessed data with wavelet analysis to extract features with an accuracy of above 90%. In another study, Wu et al. [60] presented an XGB-based technique to improve identification accuracy in power transformer failure diagnostics. The proposed technique produces a hybrid diagnostic network by merging an improved genetic algorithm (IGA) with the XGBoost. Compared to other approaches like support vector machines, this strategy dramatically improves diagnostic accuracy. As a result, the proposed approach is presented as a feasible option for detecting various forms of transformer defects.

2.3.3.3.2 Random Forests (RF)

Another tree-based machine learning technique that may be used for regression and classification is Random Forest. It also does well in dimensional reduction methods, missing values, outlier values, and other essential data exploration operations. It is an ensemble learning method that combines many weak models to create a more robust model. [61]. Random Forest creates many trees to categorize a new object based on the available attributes. Each tree generates a categorization for that class,

referred to as a "vote." When it comes to regression, the forest chooses the classification that receives the most votes (across all trees in the forest) and averages the outputs from different trees. The Random Forest algorithm is a bagging ensemble technique extension. Instead of using all features to build trees, RF uses a random subset of data and a random selection of features to train the model [47], [62].

Random Forest classifications, like XGBoost, are both accurate and efficient in terms of computation, and the results are understandable. As a result, random forest classifiers are well-suited to industrial contexts, where large datasets are not always accessible for training diagnostic models. Cerrada et al. [4] devised a reliable approach for identifying multi-class faults in spur gears. The diagnostic system uses evolutionary algorithms for feature selection and a random forest classifier in a supervised context. The approach is verified using actual vibration signals by analyzing different fault classes: an incipient fault under varied load and velocity conditions. Patel and Giri [42] also utilized a random forest classifier to diagnose multi-class mechanical problems in induction motor bearings. An accelerometer sensor was used to collect vibration signals from the bearings, and their values were obtained as statistical features to feed into the RF model. According to the findings, in terms of performance and accuracy in identifying bearing problems, this technique surpasses existing designs such as ANNs. Random forests have also been used to identify non-mechanical faults. Puggini et al. [63], for example, developed an unsupervised random forest approach to identify damaged wafers based on chemical fingerprints obtained during the plasma etching process.

2.3.3.4 K-Nearest Neighbors (KNN)

In order to generate predictions for new data, the KNN algorithm uses a majority voting method. The k-closest records from the training dataset are identified for each new record. The fundamental closest neighbor (NN) technique predicts categorization or regression for each random occurrence [64]. The value of the target property of the nearby records is used to construct a forecast for the new record. The usual range of values is a few dozens to a few hundred. The KNN algorithm can compete with the most accurate models since it offers perfect predictions. As a result, the KNN approach may be employed in situations requiring high accuracy, but a human-readable model is not required [47].

Data distribution has little bearing on fault identification for machine components using classification algorithms like k-nearest neighbor (KNN). Both healthy and flawed reference data are required for classification algorithms, which are frequently unavailable. KNN is a method for calculating the health index based on the distance between the test and reference data [65]. This approach has been used in several academic papers to detect and categorize problems. For example, Tian et al. [65] suggested employing KNN to identify bearing faults and monitor bearing degeneration in electric motors. Spectral kurtosis (SK) and cross-correlation are used to derive fault features that indicate discrete faults. Using principal component analysis (PCA) and a semi-supervised KNN distance metric, these attributes are combined to create a health index. A KNN was used in another study by Naik and Koley [66] to solve a supervised classification challenge. Using K-nearest neighbor, this paper proposes a fault detection and classification approach for AC/DC transmission lines using a doubly-fed induction generator (DFIG). The suggested KNN-based approach has been tested in various fault scenarios with varying fault

resistance, fault inception angle, and fault location. In all cases tested, the proposed technique achieves a fault detection and classification accuracy of 100.

2.3.3.5 Bayesian Networks (BN)

The Bayesian Network (BN) is a popular probabilistic graphical model that solves a range of uncertainty problems using probabilistic information representation and inference. BN allows specifying exponentially more significant probability distributions using a polynomial of probabilities. Localized tests, which are exclusively concerned with variables and their immediate causes, ensure that Bayesian models are consistent and comprehensive [47], [67].

For decades, BNs have been investigated and used in fault diagnostics as part of data-driven techniques. To construct BN-based fault detection models, a vast amount of historical data is utilized, and backward analysis utilizing various methodologies is performed to identify [5]. Muralidharan et al.'s research [68] is an outstanding example of using Bayesian Networks in IFD investigations. This work shows how to use discrete wavelet features extracted from vibration signals of healthy and problematic centrifugal pump components to diagnose defects using the Naive Bayes and Bayes net methods. Feature extraction, categorization, and classification comparison are the three essential processes in this technique. In order to discover the best wavelet for diagnosing centrifugal pump malfunctions, the classification accuracies of several discrete wavelet families were calculated and compared. Zhao et al. [69] introduced a Diagnostic Bayesian Networks (DBNs)-based approach for diagnosing 28 flaws in air handling units (AHUs), which covers the majority of common problems. The DBNs were built using data from three AHU fault detection and diagnosis (FDD) investigations, including a thorough examination of AHU FDD

techniques and fault patterns. According to the findings, the DBN-based method efficiently finds faults even when diagnostic information is confusing and restricted.

2.3.4 Deep Learning for IFD

Deep learning is a machine learning subclass that uses three or more neural network layers. A deep neural network is computer software that uses intricate algorithms to generate predictions and fix data faults. These neural networks are designed to replicate the function of the human brain by allowing it to learn from massive amounts of data. Additional hidden layers can help with DL model accuracy optimization and refining. Deep learning is being utilized to solve a number of issues, including digital assistants, voice-activated devices, and credit card fraud protection [17], [18]. The difference between standard machine learning and current deep learning is that deep neural networks analyze data differently. The word "deep" refers to the definition and arrangement of numerous attributes derived from the model's input data in these models. Deep learning may also be used to ingest and analyze unstructured data like text and photographs. This paradigm automates feature extraction to some extent, decreasing the requirement for human knowledge. While a machine learning expert develops a feature hierarchy by hand, deep learning algorithms can identify which features are necessary to complete the task at hand [33], [70].

Deep neural networks, as previously said, are made up of several layers of linked nodes, each of which improves and refines the prediction or categorization. The forward flow of calculations begun by the input data via the network is referred to as forward propagation. After that, backpropagation is used, which is a method that evaluates errors in predictions using methods like gradient descent and then updates

the function's weights and biases by traveling back through all of the layers [33], [71]. To function correctly, a neural network requires a precise set of parameters. The number of hidden layers, number of neurons in each layer, each layer's activation function, optimization algorithm, loss function, and ways to avoid underfitting or overfitting the network are architectural and computational factors. In the deep learning literature, these structural parameters are referred to as Hyper-parameters. To set the hyper-parameters, experience and a lot of trial and error are required. It is not straightforward to set hyper-parameters like learning rate, batch size, momentum, and weight decay [17], [36]. To fine-tune the network using the training data, most DL-based approaches go through numerous rounds of hyperparameter tweaking depending on the outcomes of prior rounds. As a result, applying deep learning models to each problem can be difficult and time-consuming, requiring a significant amount of work to achieve satisfying results. However, because the results frequently outperform non-DL approaches, using deep neural networks in a variety of situations for more desirable outcomes is efficient [16].

The feature extraction phase is not included in implementing deep learning models, despite the fact that data collection methods for getting the datasets necessary for deep learning architectures are still required [72]. As a result, using deep neural networks for IFD research necessitates obtaining large volumes of data to train the model and passing that data to the appropriate DL model. The remainder of this section delves into these two phases and the techniques and architectures used in the research publications that use deep learning to identify machine faults intelligently.

2.3.4.1 Big data collection

Big data is described as data that is so large, rapid, or intricate that it is difficult or impossible to process using traditional methods [73]. Obtaining and storing large amounts of data for analytics has a long history. The idea of big data acquired significant traction in the early 2000s when industry professionals established the now-mainstream definition of big data as the five V's (Volume, Variety, Veracity, Value, and Velocity). Big data analytics can help in decision-making, modeling and forecasting future occurrences, and improving business intelligence [17], [41].

In today's sector, most industrial activities follow Big Data features. For example, most manufacturing operations are performed by a group of machines, and fault diagnosis is usually centered on machine groups. As a result, throughout the long-term operation of various pieces of equipment, the monitoring system should continually gather data. As a result, the amount of information gathered tends to increase (Volume) [31]. Furthermore, while the monitoring system may capture a large quantity of data, only a tiny fraction of that data is relevant and has the appropriate value. Furthermore, multi-source sensors are used to collect various sorts of data. A monitoring system, for example, may include vibration and speed data from a condition monitoring system as well as some control parameters from supervisory control, showing a wide range of data [37]. Finally, the development of sensor technologies and data transmission has made it simpler to gather enormous volumes of data-carrying real-time information while also making it easier to monitor data streams adequately. This emphasizes the importance of data gathering systems that can sustain high Velocity [4].

2.3.4.2 DL-based intelligent fault diagnosis

Deep learning-based diagnosis systems learn features from input data and use them to detect machine health conditions. By learning feature hierarchies using features from higher levels of the hierarchy formed by the composition of lower-level features, deep learning algorithms have the potential to address the inadequacies mentioned above in present intelligent defect detection systems. [40]. By applying non-linear operations, these models leverage hierarchical networks such as multi-layered Auto-Encoders, CNN, and RNN to discover significant characteristics [74]–[76]. Through these non-linear transformations, deep learning-based techniques may adaptively acquire representation information from input signals and estimate sophisticated non-linear functions with a small error. The model learns to associate these traits with other classes in succeeding layers and provides the model's output. The output layer determines the machine's health and the sort of problem it may be experiencing [77]. Because of its high capacity for multi-class classification, an ANN-based classifier is the architecture of choice in most applications. The Backpropagation approach is used to update the training parameters of the diagnostic models after each round of training, and the error between the actual output and the target is minimized during this phase [18].

Fully-connected (FC) neural networks, also known as Dense neural networks, are the most often utilized forms of NNs in deep learning research and are still employed in most DL models. All nodes (neurons) in one layer are linked to the neurons in the next layer in this arrangement. DNN networks are deep learning workhorses that are used in tens of thousands of applications [18], [33]. On the other hand, fully-connected models cannot excel at all tasks, albeit they have been shown to produce superior outcomes when describing more complicated functions. Researchers have

also shown that a deeper network with the same number of neurons may learn more complex functions than a shallower network and that the layered structure of such networks can aid in their learning ability [16].

On the other hand, such networks need a lot of processing power and are prone to overfitting. As a result, while these networks are exceptionally broadly applicable due to their flexibility, they perform poorer than special-purpose networks tailored to a particular area's structure [70]. As a result, the majority of fault detection research articles include a fully-connected structure as part of their suggested solution, while FC-only networks are not used in the majority of studies. Another reason is that IFD datasets frequently contain signals and temporal data, which convolutional or recurrent neural networks are better at anticipating. However, researchers such as [78], [79] used an autoencoder to reduce dimensionality and passed the information to a fully-connected network to categorize the problems. The rest of the researchers used FC layers in conjunction with the other designs described in the rest of this section.

2.3.4.2.1 Auto-Encoders (AE)

An autoencoder (AE) is an unsupervised learning method for learning representations that use neural networks. Due to a network bottleneck, autoencoders are neural network designs that produce a compressed knowledge representation of the original input. An autoencoder's purpose is to train the network to capture essential bits of the input in order to build a lower-dimensional representation (encoding) for higher-dimensional data, which is commonly used to reduce dimensionality [17]. If the data has any structure (for example, correlations between input properties), that structure can be learned and exploited to drive the input

through the network's bottleneck. This network may be trained using the reconstruction error, which measures the differences between our initial input and the subsequent reconstruction [72]. A key element of network architecture is the presence of an information bottleneck. Our network may quickly learn to memorize the input values and send them via the network if there is no bottleneck. A bottleneck limits the quantity of data that can pass through the network, resulting in input data compression. Creating an autoencoder architecture is difficult to ensure that the compressed data accurately replicates the original input [80]. As a result, the research community has developed several autoencoder designs that can accomplish the encoding task in various scenarios. For example, given a corrupted form of data as input, a denoising autoencoder (DA) is trained to reconstruct/denoise the clean input x from its damaged sample. The most commonly used noise is dropout noise/binary masking noise, which randomly sets a fraction of the input attributes to zero. Several DAs may be stacked to create a deep network capable of learning representations by feeding the outputs of each layer as inputs to the next layer. Because autoencoders, particularly stacked denoising auto-encoders (SDA), may be trained unsupervised, they can provide an effective pre-training solution by initializing the weights of a deep neural network (DNN) to train the model [76]. Getting the model to acquire a meaningful and generalizable latent space representation is typically the most challenging component of dealing with these autoencoders [17], [81].

Autoencoders and their common modifications have been employed in machine fault diagnosis in various publications. Many researchers used AE versions to learn properties from sensor data automatically and subsequently fulfill machine diagnosis tasks [76]. For example, Sun et al. [79] used a sparse auto-encoder (SAE) to extract

features from a deep neural network approach for identifying induction motor defects. According to research, the SAE-based DNN outperforms ordinary neural networks in a machine error simulator. Jia et al. [78] devised another intelligent technique that uses a mix of fully-connected neural networks and an autoencoder to minimize the dimensionality of information to overcome the drawbacks of previous intelligent diagnosis systems [81]. The recommended approach is validated using data from rolling element bearings and planetary gearboxes. Compared to earlier techniques in this area, the results show that it adaptively mines accessible fault characteristics from observed signals and enhances diagnosis accuracy. Furthermore, Lu et al. presented the stacked denoising autoencoder (SDA), a deep feature learning approach that proved successful and trustworthy for IFD situations. SDA has long been a popular way of obtaining the promised benefits of deep architecture-based robust feature representations. The technique is acceptable in health state identifications for signals containing ambient noise and working condition fluctuations. In another study, Chen et al. [82] introduced a new multi-sensor data fusion strategy to improve fault diagnosis reliability. The proposed technique uses multiple two-layer sparse autoencoder (SAE) neural networks for feature fusion. Deep belief networks (DBN) for classification are trained using fused feature vectors considered machine health indicators. According to experimental data, SAE-DBN's proposed technique outperformed traditional fusion approaches in identifying machine running conditions.

2.3.4.2.2 Convolutional Neural Networks (CNN)

Convolutional neural networks, or CNNs, a form of artificial neural network prominent in computer vision, are gaining traction in a variety of domains, including

facial recognition, climate forecasting, and medical image processing [14]. During the backpropagation phase, CNN employs numerous building blocks such as convolution, pooling, and fully connected layers to learn spatial hierarchies of characteristics automatically and adaptively. Convolution and pooling layers operate together to extract features, whereas FC layers transfer retrieved characteristics into final outputs for tasks like classification [17]. A convolution layer is a component of CNN that consists of a series of mathematical operations such as convolution, a type of linear operation. In digital photographs, pixel values are stored in a 2D grid, and at each image point, a tiny grid of parameters known as a kernel, an optimizable feature extractor, is applied. CNNs are ideal for image processing since features can emerge everywhere in the picture. As one layer feeds its output into the next, extracted properties can grow hierarchically and become increasingly sophisticated [83]. A pooling layer reduces the number of learnable parameters by performing a conventional down-sampling operation on the feature maps, lowering their in-plane dimensionality and introducing translation invariance to minor shifts and distortions. Although filter size, stride, and padding are hyperparameters in pooling operations, similar to convolution operations, the pooling layers do not contain any learnable parameters. The most common pooling procedure is max pooling, which chooses patches from input feature maps, outputs the highest significant value in each patch, and discards the rest [3], [76]. The activation function of the last FC layer is frequently different from the others. The activity function requires choosing a suitable operation for the job chosen. The SoftMax function converts target class probabilities to actual output values from the final fully linked layer. Each value ranges from 0 to 1, and the sum of all values equals 1, which is an activation function used in multi-class classification issues [17], [84].

In a range of computer vision applications where the input data is often 2D, CNN models have proven useful. However, CNNs cannot handle one-dimensional signals like vibration data, which is used to diagnose machine problems. Researchers used three alternative techniques to create a CNN-based diagnostic model and obtain optimal performance [14], [85]. Signal processing techniques such as the wavelet packet, continuous wavelet transform, and dual-tree complex wavelet transforms are used to preprocess the one-dimensional input data in order to move the signals to the 2D time-frequency domain. CNN then processes the monitoring data using a two-dimensional time-frequency model [8], [86]. In the IFD study, there are several examples of CNNs being used as the classifier module. The following are some of the most important studies in this category:

For example, by claiming that gearbox vibration signals are vulnerable to the existence of a defect, Chen et al. [75] developed a convolutional neural network implementation of a deep learning technique for defect detection and classification in gearboxes. There are 12 different combinations of fundamental condition patterns in each test case, for a total of 20 test cases with different combinations of condition patterns. The accuracy gained by calculating Root Mean Squared Error (RMSE) shows that the proposed technique is highly reliable and effective in identifying industrial reciprocating equipment defects. Furthermore, vibration patterns change due to machine state faults, vibration analysis is a well-established method for rotating equipment condition monitoring. Before their research, Janssens et al. [87] discovered that automatic fault detection depended mainly on manually-engineered aspects such as the raceway's ball pass frequencies. They identified successful bearing fault detection characteristics from the data alone. This research looked at various bearing issues, including outer-raceway flaws, lubrication degradation,

healthy bearings, and rotor imbalance. The CNN-based feature-learning method outperformed the standard feature-engineering technique, which relies on manually constructed features and a random forest classifier. Gua et al. [3] proposed an upgraded algorithm-based hierarchical, learning rate adaptable CNN. The bearing-fault data samples were collected from a test rig and utilized to validate the model's validity. The approach provided good accuracy in terms of error pattern recognition and fault size estimate. Furthermore, according to the comparison, the upgraded algorithm is well suited to the fault-diagnosis model, and the recommended strategy outperforms other current techniques. Jing et al. [8] constructed a CNN that can learn feature representations from vibration signal frequency data, claiming that only a few research studies have employed deep learning in feature learning for mechanical diagnostics. They examined how well feature learning from raw data, frequency spectrum, and mixed time-frequency data was performed. Data from the PHM 2009 gearbox challenge and a planetary gearbox test rig were used to illustrate the effectiveness of the proposed technique. In another study, Wen et al. [85] propose a new CNN for defect identification based on LeNet-5. When tested on three well-known datasets: the motor bearing dataset, self-priming centrifugal pump dataset, and axial piston hydraulic pump dataset, the recommended approach achieved a prediction accuracy of 99.79 percent, 99.481 percent, and 100 percent. These results have outperformed traditional methods like SVM and deep belief networks. Several other studies, such as [12], [88], [89], examined gearboxes and rolling bearings using CNN-only models for feature extraction and classification. Compared to traditional algorithms like ANNs and SVMs, these strategies enhanced classification performance on target datasets. For predicting the remaining usable life of machine parts and their current health status, similar methodologies are used [90].

Wang et al. describe the use of CNN-based hidden Markov models (CNN–HMMs) to categorize multi-faults in mechanical systems in order to enhance feature learning [91]. The average classification accuracy ratios for two data series with acceptable error rate reductions are 98.125 percent and 98 percent, respectively. Another study uses cognitive computing theory to explore the benefits of image recognition and visual perception in bearing issue diagnoses [86]. In the temporal dimension, this CNN model reduces learning calculation needs. Identifying the essential features of bearings allows this model to operate in ambient noise with a high degree of invariance. The CNN model's efficacy for fault classification of rolling bearings was determined through contrast testing and analysis. Recent publications have proposed innovative techniques such as Deep Convolutional Neural Networks with Wide First-Layer Kernels have been proposed in recent publications (WDCNN). For instance, Zhang et al. [39] proposed a WDCNN that takes raw vibration signals as input and augments them to acquire extra information. Using large kernels in the first convolutional layer, this model collects features and suppresses high-frequency noise. Based on the frequency characteristics of standard signals, WDCNN outperforms the state-of-the-art DNN model under different working loads and noisy environmental conditions. Recent CNN-based defect detection algorithms, on the other hand, typically use transfer learning since such models require vast datasets to be trained from scratch, which would take a long time and a lot of computing power. The fourth section of this article will go over these works.

2.3.4.2.3 Recurrent Neural Networks (RNN)

A recurrent neural network (RNN) is an artificial neural network that works with data in time series or sequences. Language translation, speech recognition, natural

language processing, and picture captioning are examples of temporal or ordinal issues used by these DL methods. Ordinary feedforward neural networks are designed to handle data that are unconnected to each other [17], [20]. The neural network must be adjusted to account for these dependencies if the data is organized in a sequence where one data point relies on the previous data point. RNNs have a memory notion that allows them to remember the states or information from previous inputs to create the sequence's subsequent output [28], [92]. The fact that recurrent networks' parameters are shared across all network levels distinguishes them even more. Each node in a feedforward network has a different weight, but each recurrent neural network layer has the same weight. RNNs determine gradients using the backpropagation through time (BPTT) approach, which differs from ordinary backpropagation in that it is specialized in sequence data. The model trains itself by computing errors from its output layer to its input layer in traditional backpropagation, similar to BPTT. These calculations allow us to change and fit the model's parameters precisely. BPTT differs from typical techniques in that it totals mistakes at each time step, whereas feedforward networks do not require this [33], [74]. RNNs frequently run into two problems during this procedure: expanding gradients and disappearing gradients. These issues are defined by the magnitude of the gradient, which is the gradient of the loss function along the error curve [17], [20]. RNN variants, including Long short-term memory (LSTM) [93], Gated recurrent units (GRUs) [94], and Bidirectional recurrent neural networks (BRNN) [17], are developed to overcome these issues.

RNN-based algorithms can deal with data sequences of varying lengths and capture long-term relationships in signals received from machines. They integrate model training and representation learning into a single neural network, needing no

additional domain knowledge. Furthermore, these structures may allow for discovering previously undiscovered structures, allowing the model's generalization capabilities to be improved [76]. To improve their performance, these architectures are frequently combined with other models like autoencoders or CNNs [95]. An IFD approach based on a Gated Recurrent Unit (GRU)-based denoising autoencoder was suggested by Liu et al. [74]. This method predicts various rolling bearing vibration levels based on the preceding period for the next period. These GRU-based non-linear predictive denoising autoencoders (GRU-NP-DAEs) have good generalization capability and are trained for each defect type. Experiments show that the suggested approach has a high degree of resiliency and accuracy in categorization. Zhao et al. [95] used a combination of recurrent and convolutional neural networks to solve the fault classification problem. Convolutional Bi-directional Long Short-Term Memory (CBLSTM) networks are used in this study to deal with basic sensory information. To forecast the target value, stacked FC and linear regression layers are built on top of bi-directional LSTMs. Our model surpasses numerous state-of-the-art baseline techniques, and it can forecast based on real-world raw data, according to the results of the testing. However, robust RNNs are used as the only classifier module to detect equipment faults in certain circumstances. For example, Rafique et al. [21] suggested a novel deep learning approach based on Long short-term memory (LSTM) networks for defect detection and classification in electrical power transmission networks. The approach uses LSTM units that work directly on the operational data rather than on features to generate an end-to-end model based on the temporal sequence of the power system's operational data. End-to-end learning speeds up decision-making by learning directly from labeled datasets and eliminates the need for complex feature extraction.

The proposed approach has shown a quick reaction in time performance and is adaptable to operational scenarios.

2.3.4.2.4 Deep Belief Networks (DBN)

The Deep Belief Network (DBN) is a form of Deep Neural Network that learns a deep network structure layer by layer using layers of Restricted Boltzmann Machines (RBMs) placed on top of each other. DBNs were initially proposed as generative models, and they may be used to tackle unsupervised learning challenges by reducing feature dimensionality. They may also be used to create classification or regression models in supervised learning applications [33]. Layer-by-layer training and fine-tuning are the two steps of DBN training. Fine-tuning refers to applying error back-propagation methods to fine-tune the parameters of the DBN after the unsupervised training is done [6]. Because of its benefits, such as rapid inference and the ability to represent deeper and higher-order network topologies, this structure has recently gained appeal in machine learning [76]. As a result of these advantages, various DBN application examples in fault diagnostic research are shown below:

Tamilselvan et al. [38], for example, describe a novel multi-sensor health detection system based on deep belief networks. The proposed technique is based on DBN state classification and is a multi-sensor health diagnostic methodology with a three-step framework. Aircraft engines and electric power transformers have both been diagnosed using this technology. To demonstrate the use of the suggested methodology, benchmark concerns and two engineering health diagnosis applications are employed. Gan et al. [6] conducted another study for the hierarchical identification of mechanical systems, in which they constructed a unique

hierarchical diagnostic network (HDN) by collecting deep belief networks (DBNs) in each layer. The deeper layer in HDN categorizes the output from the preceding layer more thoroughly and gives representative characteristics for specific tasks. A two-layer HDN is developed for a two-stage diagnosis with the wavelet packet energy feature. The trials show that HDN is highly reliable for multi-stage diagnosis and that it can overcome the overlapping problem caused by noise and other disruptions.

2.3.5 Transfer learning and generative models

So far, we have discussed how machine learning and deep learning have transformed IFD research by allowing models to discover faults without the need for human specialists automatically. However, in the age of deep learning, these models require large datasets to perform well and operate correctly in real-world scenarios. The problems that DL models are designed to detect are rare compared to fault-free scenarios. As a result, deep learning models cannot complete most IFD tasks due to a lack of data. Modern deep learning research for discovering machine flaws focuses on approaches that can help even when there is insufficient labeled data. Transfer learning or generative models are commonly used to leverage pre-trained models for new challenges or to produce extra data based on previous datasets. These methods are widely regarded as the future of deep learning in many fields, and they are the most widely used approaches to furthering intelligent defect detection research. This section looks at these tactics and how they have been used in IFD research.

2.3.5.1 Transfer learning

The practice of improving learning in a new activity by transferring information from a previously mastered related task is known as transfer learning (TL). While

most machine learning algorithms are developed to solve specific problems, the development of algorithms that allow for transfer learning is a hot issue in the machine learning field. [24]. The goal of transfer learning is to increase learning in the target task by using knowledge from the source task. Transfer approaches are frequently reliant on the machine learning algorithms used to learn the tasks and might be considered extensions of such algorithms [45]. Inductive learning entails using well-known classification and inference techniques like neural networks, Bayesian networks, and Markov Logic Networks to make new discoveries [96]. For instance, training neural networks consume many resources because of the models' complexity. Transfer learning is used to improve the efficiency of the process and reduce the number of resources required. Any transferable knowledge or characteristics may be transported between networks to speed up the construction of new models. Building such a network necessitates applying knowledge across many jobs or contexts. Transferred knowledge is generally confined to broad processes or tasks that may be used in a variety of settings [97]. Transfer learning may also be employed in computer vision applications, such as recognizing and categorizing picture subjects using machine learning algorithms trained on large datasets of images. Transfer learning will be used in this case to apply the reusable characteristics of a computer vision algorithm to a new model. The exact models produced from vast training datasets may be adapted to smaller collections of photos with the help of TL. This entails translating the model's more general capabilities, such as detecting the borders of objects in photographs. After then, the model's more specific layer, which identifies various objects or forms, may be trained. The model's parameters will need to be fine-tuned and optimized, but the model's core functionality will have been developed through transfer learning [17], [30], [97].

There are three main ways that transfer might aid models in learning more successfully. In contrast to an ignorant agent's first performance, the first performance is accomplished in the target task using just transmitted knowledge before any subsequent learning. The second consideration is the time it takes to fully comprehend the target task utilizing transferred information versus learning it from scratch. The third component is the difference between the ultimate performance level reached in the target task and without transfer [7], [96].

Using transfer learning approaches, large-scale machine learning models will be adjusted for individual activities and situations. Transfer learning will help spread machine learning models across new sectors and businesses [17]. As a result, TL approaches are being used in various applications, such as intelligent defect detection. They are regarded as one of the critical components in the future of deep learning research [98]. IFD is expected to go beyond academic research and into engineering. In this situation, it is possible to simulate various problems and obtain adequate labeled data from laboratory-used bearings. The diagnostic models developed with them may be used to diagnose bearing issues in engineering contexts if the diagnosis information could be reused. Transfer learning accomplishes the aim mentioned earlier by allowing information from one or more diagnosis tasks to be used in different but related actions [29], [99].

Many research publications have presented defect diagnostic methods based on transfer learning in recent years. For example, Lu et al. [7] proposed a domain-adaptable deep neural network model that used transfer learning while simultaneously boosting the original data's representative information, resulting in high classification accuracy in the target domain. They presented many methods for identifying the appropriate hyperparameters for the transferred model. The Deep

Adaptation in Fault Diagnosis (DAFD) model is proposed in this paper to solve cross-domain learning issues in fault diagnosis. DAFD's purpose is to learn transferable features that bridge the cross-domain gap while maintaining the recognized information in the original data. The utility and reliability of both the recommended model and the exploring approaches for the parameters in this study were shown by experimental findings on real-world datasets. The DAFD model may include existing deep neural network architectures such as DBN and CNN. A deep transfer learning model called DTL was created by Wen and Gao [99] in another study for fault diagnosis. To extract raw data properties, this technique uses a three-layer sparse auto-encoder and the highest mean discrepancy term to minimize the discrepancy penalty between training and testing data features. The proposed TL model was tested using the well-known motor bearing dataset from Case Western Reserve University. In most investigations, DTL achieved higher prediction accuracies than DL, indicating a considerable improvement. DTL outperforms other algorithms, including DBN, sparse filters, SVMs, and ANNs, with a prediction accuracy of 99.82 percent. Transfer learning was also employed by Shao et al. [1] to develop a new deep learning IFD methodology that is more accurate and faster to train than previous approaches. Using a Wavelet transformation, raw sensor data was transformed into images in order to get moment distributions, and lower-level properties were recovered using a pre-trained network. The annotated time-frequency images were then used to fine-tune the neural network architecture at a higher level using the annotated time-frequency images. This paper creates a machine defect detection pipeline and tests it on three mechanical datasets: induction motors, gearboxes, and bearings, to show its usefulness and universality. The majority of datasets show test accuracy close to 100%. Gua et al. [23] proposed the

deep convolutional transfer learning network (DCTLN) in another study. The two components that make up this technique are condition recognition and domain adaptation. The condition detection module uses a one-dimensional neural network, which automatically learns properties and detects machine health conditions. The domain adaptation module supports the 1-D CNN in learning domain-invariant features by increasing domain recognition errors and decreasing probability distribution distance. Six transfer fault diagnosis tests were conducted to confirm the effectiveness of the proposed approach. Ultimately, Yang et al. [100] suggested a feature-based transfer neural network (FTNN) predict the health states of bearings in real-world machines using diagnostic knowledge from bearings in laboratory machines (BLMs) (BRMs). In order to extract movable properties from natural vibration data from BLMs and BRMs, the suggested approach uses a CNN. Then, to restrict the parameters of CNN, regularization terms of multi-layer domain adaptation and pseudo-label learning are created to reduce the distribution discrepancy and among-class distance of the obtained transferable features. The proposed method can learn transferable features that may be utilized to connect BLM and BRM data. As a result, it is more accurate in diagnosing BRMs than earlier approaches.

2.3.5.2 Generative models

A generative model describes how a dataset is created in terms of a probabilistic model. By sampling from this model, users may generate new data [17]. Assuming that a collection of dog images exists, we can build a model that can construct a new image of a dog that has never existed but appears legitimate since the model has learned the general rules that influence a dog's look. This is the kind of issue that

generative modeling can help with [30]. The main goal is to create a model that can generate distinct sets of characteristics that appear to be created using the same principles as the original data. Furthermore, a generative model must be probabilistic rather than deterministic. If our model is merely a fixed computation, such as finding the average value of each pixel in the dataset, it is not generative. In this method, the model consistently produces the same outcome. To change the individual samples generated by the model, a stochastic (random) element must be incorporated [30], [101]. To put it another way, we can think that some unknown probability distribution explains why some inputs are more likely to be found in the training dataset than others. Our objective is to build a model that looks as close to this distribution as feasible, then sample from it to generate new, distinct observations that seem like they came from the original training set [98].

To understand why generative modeling may be regarded as the next frontier for machine learning, we must first evaluate why discriminative modeling has been the driving force behind the majority of developments in machine learning approaches over the preceding decades. While discriminative modeling has provided the majority of the motivation for machine learning successes, innovative deep learning applications to generative modeling difficulties have resulted in many exciting discoveries in the field in recent years [30]. From an academic viewpoint, progress in discriminative modeling is arguably easier to evaluate because performance data can be compared to specific high-profile classification tasks to determine the current best-in-class approach. Generic models are notoriously difficult to evaluate, especially when the quality of the output is primarily subjective. As a result, much work has been put into training discriminative models to attain human performance in various photo and text classification tasks in recent years [102], [103].

In recent years, IFD researchers have been particularly interested in a sort of generative model known as Generative Adversarial Networks, or GANs [30]. GANs are an intelligent way to train a generative model by framing the issue as a supervised technique with two different sub-models: the generator model, which is taught to produce new instances, and the discriminator model, which attempts to categorize examples as genuine or fraudulent (generated) [17], [104]. Both models are trained in an adversarial zero-sum situation until the discriminator model is misled around half of the time, suggesting that the generator model offers believable examples. This section examines two research publications to introduce the most prominent IFD efforts that use generative adversarial networks. Firstly, Yin et al. [104] proposed a data generation method based on the Wasserstein generative and convolutional neural network (WG-CNN), which used a generator and discriminator for confronting training, grew a small sample set into a high-quality dataset and used a 1D-CNN to learn sample properties and categorize different fault classes. With 100 percent classification accuracy, the proposed approach delivers an obvious and satisfactory fault diagnostic impact for few-shot learning. This method also works effectively in a variety of loud environments. Pan et al. [98] present a semi-supervised multi-scale convolutional GAN for bearing fault diagnostics with sufficient unlabeled data for training. The discriminator is a multi-scale 1D-CNN, while the generator is a multi-scale deconvolutional neural network. The model is trained using an adversarial approach. The proposed method was tested on three datasets, with classification accuracy averaging 100 percent, 99.28 percent, and 96.58 percent, respectively. According to the results, when the labeled data is insufficient, the suggested model adequately finds bearing problems.

Chapter 3

3 Methodology

3.1 Workflow

The proposed methodology of the current thesis is split into four main steps, including the data pre-processing, estimating features' importance, ML-based model development, and model assessment based on evaluation criteria. The first three steps are explained in this section as well as the utilized dataset, while the next section is dedicated to the fourth step for a thorough examination of the results and discussion.

The PHM's dataset is chosen as the fault detection source in machinery, taken from a challenge of the same name. The attendees are invited to solve a classification issue for a genuine manufacturing line employing state-of-the-art algorithms and models in this challenge. The dataset is first pre-processed to handle missing parts and is reorganized for the ML models to be interpretable. Through multiple rounds of machine learning training and testing, PHM's features are analyzed and narrowed down to more influencing ones in the feature importance procedure. Thereafter, the final model is trained based on the processed dataset, and other algorithms are also implemented for comparison. Multiple evaluation criteria are described for

comparison objectives, and finally, it is discussed how the proposed model outperforms similar algorithms.

3.2 Software

3.2.1 Python

Python is a dynamically semantic, interpreted, object-oriented high-level programming language. Its high-level built-in data structures, together with dynamic typing and dynamic binding, make it ideal for Rapid Application Development and as a scripting or glue language for connecting existing components. Python's concise, easy-to-learn syntax promotes readability, which lowers software maintenance costs. Modules and packages are supported by Python, which facilitates program modularity and code reuse. The Python interpreter and its substantial standard library are free to download and distribute in source or binary form for all major platforms [71]. Python version 3.7.6 is utilized in this research to implement the proposed method and the baseline algorithms. Numerous Python libraries are also utilized in the implementation process. For instance, the Keras library running on the TensorFlow backend is employed for deep learning purposes, which features frequently used neural-network building elements, including layers, activation functions, and optimizers. Numpy is another Python library that supports massive, multi-dimensional arrays and matrices and a wide variety of high-level mathematical functions to manipulate such arrays. Pandas is a data manipulation and analysis software package that couples with the Numpy library and is utilized for data storage and processing. It includes data structures and methods for manipulating numerical tables and time series [71]. In Python, Scikit-learn (Sklearn) is among the most

efficient and robust machine learning packages. It uses a Python consistency interface to provide swift tools for machine learning and statistical modeling, such as classification, regression, clustering, and dimensionality reduction. This library is the basis of ML implementations of the proposed method and several other baselines and pre-processing algorithms [32].

3.2.2 Jupyter Notebook

The Jupyter Notebook App is a web-based server-client application for editing and executing notebook papers. The Jupyter Notebook App can be run locally on a computer without internet access or remotely on a server and accessible through the internet. The Jupyter Notebook App contains a "Dashboard" (Notebook Dashboard), a "control panel" that shows local files and allows opening notebook papers or shutting down their kernels, in addition to displaying, editing, and running notebook documents. The Jupyter Notebook is an excellent tool for generating and interactively presenting data science projects. The notebooks combine graphics, narrative writing, mathematical calculations, and other rich media with code and output in a single document. To put it another way, it provides a single page where code can be run, the results can be seen, and explanations, formulae, and charts can be added to make work more precise, repeatable, and shared [17], [71]. Jupyter Notebook version 6.0.3 is used in this research.

3.2.3 Google Colab

Google Colab was created to give anyone who requires GPUs or TPUs to build a machine learning or deep learning model free access to them. Google Colab may be thought of as a more advanced version of Jupyter Notebook. Colab notebooks let users blend executable code and rich text, as well as graphics, HTML, LaTeX, and

more, in a single document. Colab notebooks are saved in Google Drive accounts when created. They can be quickly shared on Colab notebooks with coworkers or acquaintances, enabling them to provide comments or make changes [105]. A free version of Colab Notebooks was utilized for deep learning implementations of the baseline algorithms.

3.3 Dataset

In this experiment, the dataset from the PHM challenge[1] [106] is used to test machine learning architectures. This dataset has been given exclusive access to extensive datasets created from a real-world industrial testbed (CSEM) in collaboration with the Swiss Center for Electronics and Microtechnology. Conveyor belt motors, an infrared camera, and robotic arms are all part of the system, allowing for continuous electrical components testing. Data was gathered in error-free working environments and under controlled conditions using a variety of seeded flaws with the assistance of subject experts. A faulty system in this case study might result in components being rejected needlessly, the testing tempo decreasing, or the testing phase switching.

The experimental dataset contains 50 signals, each describing the evolution of a variable of interest across time. Depending on the experiment, it might take one to three hours. Environment monitoring signals (such as temperature and humidity), machine health monitoring signals (such as pressure and vacuum), and other variables such as ProcessMemoryConsumption and CPUTemperature are all

[1] https://phm-europe.org/data-challenge

examples. Each signal is associated with fields that define different signal attributes collected from that signal using the automated data acquisition approach [106].

3.4 Data Pre-processing

The process of converting raw data into a comprehensible format is known as data preprocessing. Machine learning algorithms cannot deal with raw data; thus, this is a critical stage in data mining, and it is required to ensure the data is of satisfactory quality [107].

In this research, the pre-processing includes multiple stages. Firstly, some valueless columns are deleted based on the researchers' experience. These data columns are useless for the fault classification task and only provide variables set in the data collection phase. Next, the NaN (Not a Number) values are handled by replacing them with zero. While such data points can be filled with other values such as the mean, minimum, and median of their columns, zero can be a suitable choice in this case. Since these values are not recorded, they should not influence the results much, and therefore, replacing them with zero is the most reasonable alternative. It is noteworthy that missing values are a regular occurrence in many real-world datasets and can skew the results or degrade the model's accuracy in machine learning models if they remain unhandled. Therefore, such data points should be deleted entirely or regulated with predefined values to maintain the model's performance [47].

In the next step, the dataset is split into two parts to separate the input features and the corresponding outputs. The input data is also transformed with the Quantile Transformer function, so the characteristics are transformed into a uniform or normal distribution. As a result, this transformation tends to spread out the most common

values for a particular characteristic. It also lessens the influence of (marginal) outliers, making it a reliable pre-processing method. Finally, the dataset is split into two groups of train and test sets with an 80/20 ratio. The former set is utilized for training the proposed algorithm and the baselines, while the test set is not observed in this phase. The algorithms are then assessed based on their performance on the test set.

3.5 Feature Importance

The term "feature importance" refers to a set of strategies for allocating scores to input features in a predictive model, indicating the relative significance of each item when producing a prediction. For issues involving forecasting a numerical value, called regression, and problems involving predicting a class label, called classification, feature significance scores can be generated. The ratings are beneficial and may be applied to a variety of circumstances in a predictive modeling challenge, including better data interpretation, gaining a better knowledge of a model, and the reduced number of input features [108].

The relevance of features in a dataset may be used to get insight into it. The relative ratings can reveal which aspects are most important to the target and, conversely, which features are least important. A domain expert might analyze this and utilize it as a starting point for obtaining more or different data. The model may be deciphered using feature significance ratings. Moreover, building a model is quite different from comprehending the data that goes into the model. Feature importance helps to understand the relationship between the characteristics and the target variable. It also aids in determining which properties are unimportant to the model. We may lower

the dimensionality of the model by using the scores generated from feature significance while training it. Higher scores are typically maintained, whereas lower values are usually removed since they are unimportant to the model. This simplifies the model, speeds up its operation, and boosts the model's overall performance. Feature Importance can also help comprehend and communicate the model to other parties. We may identify which characteristics contribute the most to the model's prediction capability by computing scores for each feature [42], [45].

A predictive model that has been fitted to the dataset is used to generate the majority of essential ratings. When creating a forecast, checking the significance score offers insight into that specific model and which elements are essential and least significant to the model. For those models that support it, this is a sort of model interpretation that can be done. However, some models do not support feature importance functions and cannot be apprehended with this method [108].

The feature importance calculations of this research are performed using scikit-learn's built-in functions. It is noteworthy that deep learning architectures work similarly to a black box and do not reveal which features are more significant in their results. Therefore, no feature importance calculations were performed for the deep neural network designed as one of the baseline methods.

The other methods, including the proposed model and the baseline approaches, were trained once using the entire dataset. Thereafter, feature importance functions were applied, and the top 15 features in their results were saved for each model. Subsequently, the algorithms were retrained utilizing datasets containing only the 15 top features. It is noteworthy that the remaining attributes were different for each algorithm, and therefore, the importance metrics were computed per model separately. Examinations revealed that this retraining helps improve models'

performance in terms of evaluation metrics since ineffective data causes bias that disrupts the final results in machine learning models.

3.5.1 SHAP Feature Importance

The SHAP (SHapley Additive exPlanations) value is a novel feature importance calculation method known as a genuine game-changer in machine learning interpretation. Both regression and classification problems can benefit from the SHAP value, which works on various machine learning models, including logistic regression, SVM, tree-based models, and neural networks. Even if the features are linked, the SHAP value can assign the feature priority appropriately in a regression situation [109].

SHAP's purpose is to compute the contribution of each feature to the prediction of an instance x in order to explain it. Shapley values are computed using the SHAP explanation technique based on coalitional game theory. A data instance's feature values operate as coalition members. Shapley values provide us with how to distribute the "payout" (= prediction) among the characteristics in a fair manner. For example, a player might be a single feature value with tabular data. A player can also be a collection of different feature values. Pixels, for example, can be grouped into superpixels, and the prediction is spread among them to describe a picture [42], [109].

SHAP has been identified as a better alternative to compute feature importance after confirming the positive effect of the feature's importance calculation and retraining with a limited number of significant features. This research takes advantage of the SHAP library to estimate Shapley Additive Explanations values. This process is also

performed on all algorithms except for the deep learning baseline model and determines the top 15 influential features for retraining.

3.6 Proposed Algorithm

Initial examinations in this research have proved that tree-based machine learning algorithms, such as XGBoost and Random Forests, are able to outperform similar models. Artificial neural networks and deep learning architectures also demonstrated promising results with greater computational overload. Therefore, using tree-based algorithms is beneficial in terms of accuracy and evaluation metrics and being computationally optimal. Consequently, multiple tree-based machine learning models were trained and tested on the PHM's preprocessed dataset to discover the best-performing algorithm. These investigations were made to optimize the accuracy of the proposed method as well as the training time and hardware resources required. The results revealed that the Extra Tree classifier outperforms all other approaches, and therefore, it is selected as the proposed method presented in this thesis.

Extra Trees (ET), or Extremely Randomized Trees, is an ensemble machine learning technique. It is a decision tree ensemble similar to other methods such as bootstrap aggregation (bagging) and random forest. The Extra Trees approach utilizes the training dataset to generate a considerable number of unpruned decision trees. In the case of regression, predictions are formed by averaging the forecast of the decision trees, whereas, in the case of classification, majority voting is used [110].

The Extra-Trees technique creates an ensemble of unpruned decision trees or regression trees according to the standard top-down process. Its two primary distinctions from previous tree-based ensemble approaches are dividing nodes at

random and growing trees using the entire learning sample (rather than a bootstrap replica). Unlike bagging and random forest, which build each decision tree using a bootstrap sample of the training dataset, Extra Trees fit each decision tree to the whole training dataset. Similar to the random forest, this technique will sample characteristics at each split point of a decision tree at random. Unlike a random forest, which chooses an ideal split point using a greedy algorithm, the ET approach chooses a split point at random [111], [112]. Figure 3-1 represents the workflow of an Extra Tree classifier.

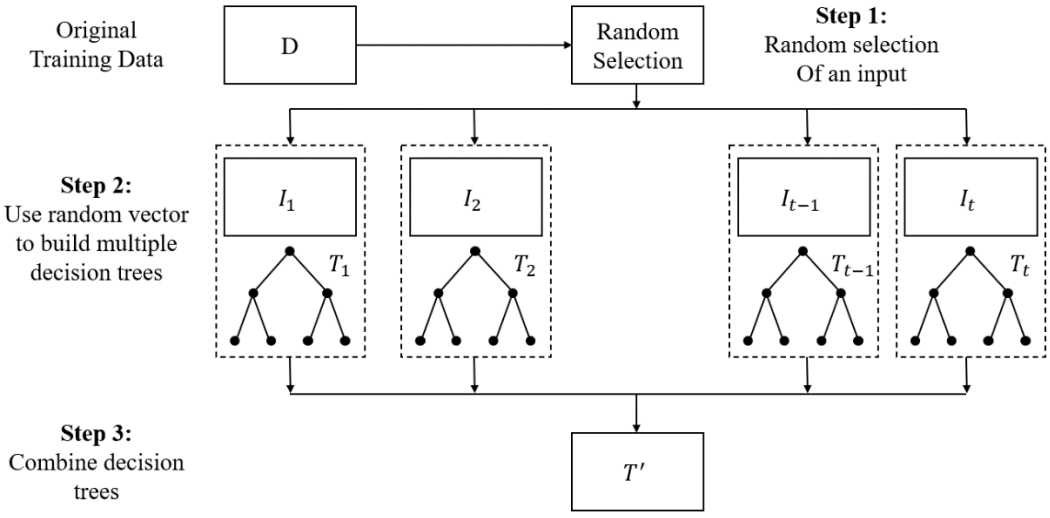


Figure 3-1 An Extra Tree classifier's visual workflow representation [112]

The number of decision trees in the ensemble, the number of input features to choose and examine for each split point randomly, and the minimum number of samples necessary in a node to establish a new split point are the three significant hyperparameters to adjust in the method. *K* is the number of randomly picked characteristics at each node, and *nmin* is the minimum sample size for splitting a node. *M* denotes the number of trees in this ensemble. The parameters *K*, *nmin*, and *M* have diverse effects: *K* controls the strength of the attribute selection process,

nmin controls the strength of averaging output noise, and *M* controls the strength of the ensemble model aggregation's variance reduction. The algorithm's variance is increased by the random selection of split points, which makes the decision trees in the ensemble less correlated. By increasing the number of trees in the ensemble, this increase in variance may be mitigated [42], [110].

The implementation of the proposed Extra Tree algorithm is executed using the scikit-learn library. The parameters set for this algorithm are n_estimators=100 which specifies the number of trees in the forest, criterion=gini, which denotes the function to calculate the splits' quality, and min_samples_split=2 showing the minimum number of samples demanded to split an inner node. The rest of the parameters are set to default values in this implementation. A Bayesian optimization algorithm enhanced these parameters for the best results on the PHM's dataset. A Bayesian Optimization is a systematic approach based on the Bayes Theorem for directing an efficient and adequate search of a global optimization issue. It works by creating a surrogate function, a probabilistic model of the objective function that is then efficiently searched with an acquisition function before candidate samples are picked to assess the genuine objective function. Bayesian Optimization is used to optimize the model's hyperparameters using the validation dataset, consisting of 20% of the training set.

3.7 Baseline Algorithms

3.7.1 XGBoost

The XGBoost algorithm has been shown to be highly versatile in many learning contexts, faster than gradient boosting, and allows regularization approaches. It also

employs parallel processing to provide faster outcomes in real-time situations. In comparison to deep learning architectures, it is also a good algorithm for small to medium datasets.

The Python programming language and the XGBClassifier module of the "xgboost" library were used to create this classifier. This module aims to do a multi-class classification using the dataset's input parameters. It classifies the findings using a sigmoid function (equation 3-1), assigning a probability value in the range of 0 to 1 for each combination of inputs and all accessible classes. According to the model, the input data is predicted to belong to the class with the highest probability value. The gbtree booster is also chosen as the classifier's booster core, and its learning rate is adjusted to 0.3 after several tests with different values.

$$Sigmoid(x) = \frac{1}{1 + e^{-x}}$$

Equation 3-1

The dataset is subjected to further preprocessing and development steps. Firstly, the dataset is encoded and scaled using several approaches in order to determine the optimum form of feature representation. The AutoML python libraries are used in this procedure. AutoML techniques are ways for automatically and quickly finding a high-performing machine learning model pipeline for a predictive modeling challenge. The major AutoML libraries for Scikit-Learn used in this study were Hyperopt-Sklearn and TPOT. The results showed that encoding techniques did not considerably enhance classification quality, and as a result, they are not used in the primary implementations to minimize model time complexity. After this stage, influential characteristics are extracted to prepare the dataset for machine learning models.

The dataset is subjected to univariate feature selection, which uses univariate statistical tests to determine the best features. It may be regarded as a second round of preprocessing before employing a Scikit-learn estimator. Feature selection processes are exposed as transform method objects in Scikit-learn. One of the strategies employed for this study is SelectKBest, which eliminates everything except the K highest-scoring parts. The remaining characteristics are then incorporated into machine learning models to categorize faults into one of nine categories.

3.7.2 CATBoost

Yandex's CatBoost machine learning algorithm was recently open-sourced. It is simple to interface with deep learning frameworks such as TensorFlow from Google and Core ML from Apple. It can work with a variety of data formats to assist organizations in addressing a variety of challenges. To top it off, it has the highest accuracy in the industry. It is enticing in two ways: It provides robust out-of-the-box support for the more descriptive data formats that accompany many business challenges. It produces state-of-the-art outcomes without the substantial data training required by other machine learning approaches [113].

The term "CatBoost" is derived from the phrases "Category" and "Boosting." As previously stated, the library works well with a variety of data types, including audio, text, picture, and historical data. Because this library is based on the gradient boosting library, the name "Boost" derives from the gradient boosting machine learning technique. *Gradient boosting* is a powerful machine learning approach that has been used to solve a variety of commercial problems, including fraud detection, recommendation items, and forecasting. It can also produce excellent results with a

small amount of data, as opposed to DL models, which require a large amount of data to train from [113], [114].

CatBoost produces cutting-edge results that compete with any major machine learning algorithm in terms of performance. We may utilize CatBoost without any explicit pre-processing to convert categories to numbers. CatBoost translates categorical data to numerical values using a variety of statistics based on categorical characteristics and categorical and numerical features. It eliminates the need for intensive hyper-parameter adjustment and decreases the risk of overfitting, resulting in more generic models. However, it includes a variety of parameters to tweak, including the number of trees, learning rate, regularization, tree depth, fold size, bagging temperature, and others [114].

The implementation of this model is performed using the Python programming language and its "catboost" library, which contains a CatBoostClassifier function. The parameters are all set to default values in this implementation since other sets of hyper-parameters showed unsatisfactory results.

3.7.3 Hist Gradient Boosting Classifier

As mentioned earlier, gradient boosting is a machine learning technique that uses an ensemble approach. Boosting is an ensemble learning technique that sequentially adds tree models to an ensemble. Each tree model that is introduced to the ensemble tries to correct the prediction mistakes generated by the tree models that are already available. Gradient boosting is a statistical framework that extends the capabilities of boosting algorithms such as AdaBoost by treating the training process as an additive model and allowing the use of arbitrary loss functions. As a result, for most

structured (e.g., tabular data) predictive modeling problems, gradient boosting ensembles are the preferred approach [113].

Although gradient boosting works effectively in reality, it might take a long time to train the models. This is because, unlike other ensemble models such as random forests, trees must be built and added sequentially, whereas ensemble members may be trained in parallel, utilizing multiple CPU cores. As a result, much work has gone into developing strategies to increase the gradient boosting training algorithm's efficiency. Extreme Gradient Boosting (XGBoost) and Light Gradient Boosting Machines (LightGBM) are two important libraries that include numerous recent efficiency strategies for training gradient boosting algorithms. The creation of each decision tree, whose speed is limited by the number of instances (rows) and features (columns) in the training dataset, is one part of the training method that may be expedited [115]. Large datasets, such as those with tens of thousands of samples or more, can make tree building particularly slow since split points on each value for each attribute must be examined. Reducing the number of values for continuous input characteristics can considerably speed up the development of decision trees. This is accomplished by discretizing or binning values into a set of buckets. The number of unique values for each characteristic may be reduced from tens of thousands to a few hundred. This enables the decision tree to work with ordinal buckets (integers) rather than particular values in the training dataset. This imprecise approximation of the input data often has little or no influence on model skill, if any, and drastically speeds up the creation of the decision tree [116].

Furthermore, efficient data structures can be utilized to describe the binning of the input data; for example, histograms can be employed, and the tree construction method can be further tuned to make effective use of histograms in the tree

construction. These approaches were created in the late 1990s to speed up the development of single decision trees on massive datasets. However, they may also be utilized in decision tree ensembles, such as gradient boosting. As a result, a gradient boosting technique that supports "histograms" in current machine learning libraries is commonly referred to as histogram-based gradient boosting [117].

A histogram is a visual representation of the frequency of data (number of occurrences) over discrete time intervals called bins. The histogram approach is theoretically straightforward, and each bin reflects the frequency of the corresponding pixel value [116], [117].

This thesis uses scikit-learn's implementation of the histogram-based gradient boosting algorithm. There are various parameters for fine-tuning the specialized algorithm to produce the best outcomes in general for all classes. Learning rate, max iter, max depth, and l2 regularization are essential parameters for the HBG classifier. Learning rate deals with shrinkage, max_iter with the number of iterations required to get a good result, and max_depth with multiple trees (Decision tree concepts). The loss function is also set to Categorical Crossentropy, which is one of the best choices for multi-label classification problems.

3.7.4 Deep Neural Network

A deep neural network is employed to perform the classification as the representative of deep learning methods. However, many preprocessing steps are skipped for the deep learning approaches, and the DL models are provided with all features to extract for themselves. The feature importance process is also not performed for this implementation since deep learning models are powerful at discovering influential features and ignoring the insignificant ones. As a result, the entire data frame is fed

to the designed neural network, trained for several epochs, and does not require the two-step feature extraction and retraining process performed for the other algorithms. Nevertheless, since random weight initializations can affect deep learning algorithms' results in many cases, the training process is repeated ten times to eliminate the effect of randomness. The models are trained separately and aggregated by averaging the test results on each metric. It is noteworthy that none of the model's hyperparameters are changed in this process. The train and test sets are also randomly picked with an 80/20 ratio for each training step.

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 dense (Dense)                (None, 50)                8600

 dense_1 (Dense)              (None, 100)               5100

 dense_2 (Dense)              (None, 200)               20200

 dense_3 (Dense)              (None, 300)               60300

 dense_4 (Dense)              (None, 200)               60200

 dense_5 (Dense)              (None, 100)               20100

 dense_6 (Dense)              (None, 50)                5050

 dense_7 (Dense)              (None, 9)                 459

=================================================================
Total params: 180,009
Trainable params: 180,009
Non-trainable params: 0
_________________________________________________________________
```

Figure 3-2 Summary of the deep neural network architecture and trainable parameters.

The deep learning predictor used in this experiment is a Fully-Connected (FC) network without RNN or CNN layers. It is implemented using the Python Keras library and the TensorFlow deep learning engine and is built employing a Sequential

model architecture [118]. These architectures perform adequately in the presence of signals or temporal variables in the dataset; however, our dataset does not contain such features. Therefore, an FC neural network seems to be suitable for experimental purposes. This network contains seven hidden layers with more than 180 thousand trainable parameters and uses an output layer of 9 neurons with a SoftMax activation function to distinguish various classes (Equation 3-2). Figure 3-2 demonstrates a summary of the deep learning architecture. The hidden layers contain 50 to 300 neurons, use the ReLU activation function (Equation 3-3), and are optimized using the RMSprop function with a learning rate of 0.0001 and a momentum of 0.1. A Sparse Categorical Crossentropy loss function is chosen for the model to calculate the model's error for optimization purposes. An early stopping mechanism is also defined with the patience of 20 epochs to monitor the loss function on the validation set and stop the training process if no practical improvement is achieved. Model checkpoints are also defined to save the best version of the model based on its performance on the validation set. Finally, the model is trained using the assembled dataset in each step, in which the training batch is also split into two parts for producing a separate validation set. This process can last for a maximum of 150 epochs in cases where the early stopping mechanism does not eliminate the training. Figure 3-3 illustrates the training process in one of the ten principal training phases.

$$SoftMax\ (\sigma(\vec{z})_i)\ =\ \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_i}}$$

3 Equation 3-2

$$ReLU(x) = \max(0, x)$$

Equation 3-3

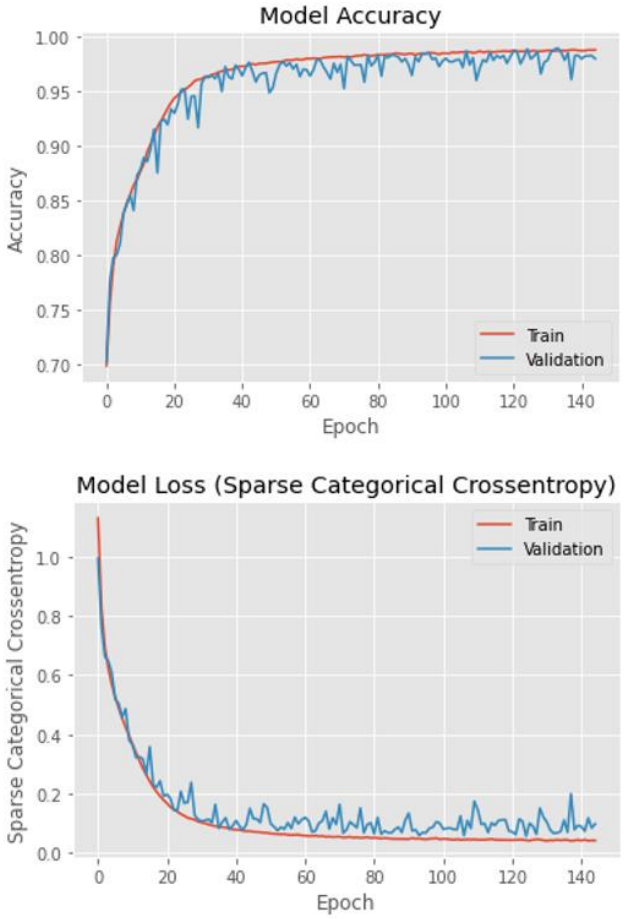


Figure 3-3 An example of the training process showing the loss function and accuracy results on the validation set in each epoch.

Chapter 4

4 Results and Discussion

After the implementations are done using the proposed method and the baseline algorithms, the evaluation procedure can assess the models' performance. In this section, the evaluation criteria used for such assessments are introduced, and the performance of each model is estimated using these metrics. Ultimately, it is discussed how the proposed method outperforms the baselines regarding accuracy measures and computational costs.

4.1 Evaluation Criteria

For this experiment, the preprocessed dataset in the previous section is separated into two sets of training and test with an 80/20 ratio. The training set is then presented to each model, allowing them to learn the features' attributes and map them to different classes. The models are not exposed to test sets until the training phase is completed. Following that, each model is given the test set as input and is expected to predict the result of each input sample based on the training data. These predictions are compared to the actual results for each combination of inputs, and the performance of each model is assessed. The evaluation metrics are common measures used to assess the classification execution of ML models: accuracy, precision, recall, the F1 score, kappa, ROC, and MATTEW.

4.1.1 Accuracy

Classification accuracy is the most frequent parameter to assess a classification prediction model's performance. Because a predictive model's accuracy is often high (over 90%), it is usual to characterize a model's performance in terms of its error rate. The first step in improving classification accuracy is to create a forecast for each sample in a test dataset using a classification model. The predicted labels are then compared to the known labels for the test set examples. The proportion of examples in the test set that was successfully predicted, divided by all predictions made on the test set, is used to determine accuracy [119].

$$Accuracy = \frac{Number\ of\ corrects\ predictions}{Total\ number\ of\ predictions}$$

Equation 4-1

The confusion matrix is another valuable method of thinking about accuracy. A confusion matrix is a table that organizes the predictions provided by a classification model by class. Each column in the table reflects the anticipated class, whereas each row represents the actual class. The number of predictions made for a class that is really for that class is represented by a value in the cell. Correct predictions are shown by cells on the diagonal, where a predicted and anticipated class align. The confusion matrix reveals not just a predictive model's accuracy but also which classes are successfully predicted, which are wrongly forecasted, and what kind of errors are being produced. A two-class classification issue with negative (class 0) and positive (class 1) classes has the most straightforward confusion matrix [120]. Each cell in this form of confusion matrix has a distinct and well-known name, which may be stated as follows:

Table 4-1 Simple confusion matrix for a binary classification

	Positive Prediction	**Negative Prediction**
Positive Class	True Positive (TP)	False Negative (FN)
Negative Class	False Positive (FP)	True Negative (TN)

From this confusion matrix, the classification accuracy may be computed as the total of accurate cells (true positives and true negatives) divided by all cells in the table.

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

Equation 4-2

While accuracy can generate a model's overall estimation, accuracy is insufficient in many circumstances, such as when the dataset is unbalanced, and not all classes have the same number of samples. In these cases, we employ alternative metrics, including precision, recall, and F1 score.

4.1.2 Precision and Recall

Precision and recall are two metrics used to measure the performance of categorization or information retrieval systems when they are added together. The percentage of relevant instances among all retrieved instances is defined as Precision, and the proportion of recovered occurrences among all relevant examples is known as recall or sensitivity. Precision and recall are both equal in a perfect classifier.

More specifically, Precision is a classification model's ability to identify only relevant data items. It is defined as the number of true positives divided by the total

number of true positives + false positives. On the other hand, recall is a model's capacity to locate all relevant examples within a data collection. The number of true positives divided by the number of true positives plus the number of false negatives defines recall in mathematics [121], [122].

$$Precision = \frac{TP}{TP + FP}$$

Equation 4-3

$$Recall = \frac{TP}{TP + FN}$$

Equation 4-4

4.1.3 F1 Score

To thoroughly assess a model's efficacy, we must examine both precision and recall. Unfortunately, precision and recall occasionally have conflicts. In other words, increasing accuracy usually decreases recall and vice versa. If the model has to recall everything, it will keep producing inaccurate outcomes, diminishing its precision. There are several instances in which both precision and recall are critical, and we do not wish to sacrifice one for the other. In such cases, an accumulative metric can solve the concerns [123].

The F1-score takes the harmonic mean of a classifier's accuracy and recall to create a single measure. It is mainly used to compare the results of two different classifiers. Assume that classifier A has a greater recall and precision than classifier B. The F1 scores for both classifiers may be used to identify which delivers superior results in this scenario. An F-score can have a maximum value of 1.0, indicating perfect precision and recall, and a minimum value of 0 if neither precision nor recall is zero [124]. A classification model's F1-score is calculated as follows:

$$F1\ Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Equation 4-5

4.1.4 Kappa

Cohen's Kappa coefficient is a statistic of interrater agreement that evaluates the degree of agreement between two variables. It is most commonly employed with data that are the product of a judgment rather than a measurement. The likelihood of agreement is compared to what would be anticipated if the ratings were independent. The range values are -1 to 1, with 1 denoting total agreement and 0 denoting complete independence. When a statistic is negative, the agreement is poorer than random. However, the definition of what constitutes an acceptable kappa value is subjective [125], [126]. The following equation shows how kappa is calculated, in which *observed agreement* is the ratio of observed agreements and *chance agreement* is the proportion of agreements expected by chance:

$$Kappa = \frac{observed\ agreement - \ chance\ agreement}{1 \ - \ chance\ agreement}$$

Equation 4-6

4.1.5 ROC

Forecasting probabilities of an observation belonging to each class rather than explicitly predicting classes might be more flexible in a classification task. This flexibility stems from the way multiple thresholds may interpret probabilities, allowing the model's operator to trade off concerns about the model's faults, such as the number of false positives vs. false negatives. This is necessary when employing models where the cost of one error surpasses the cost of other types of errors. ROC Curves and Precision-Recall Curves are two diagnostic tools that aid in

understanding probabilistic forecasts for classification predictive modeling issues [126].

A receiver operating characteristic (ROC) curve is defined as a graph that shows how well a classification model performs across all classification thresholds. The True Positive Rate (TPR) and False Positive Rate (FPR) are plotted on this graph: Precision and Recall, respectively. TPR vs. FPR at various categorization criteria is plotted on a ROC curve. As the classification threshold is lowered, more items are classified as positive, increasing both False Positives and True Positives [123]. The roc_auc_score function from the scikit-learn library's metrics module is used to calculate this score in the current research, which calculates the area underneath the Receiver Operating Characteristic Curve (ROC AUC) from prediction scores.

4.1.6 Matthew's correlation coefficient (MCC)

Accuracy and F1 scores computed on confusion matrices have been among the most popular adopted metrics in binary classification tasks. However, these statistical measures can show overoptimistic inflated results, especially on imbalanced datasets. The Matthews correlation coefficient (MCC), instead, is known as a more reliable statistical rate that produces a high score only if the prediction obtained good results in all of the four confusion matrix categories (true positives, false negatives, true negatives, and false positives), proportionally both to the size of positive elements and the size of harmful elements in the dataset [127].

$$MCC = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP) * (TP + FN) * (TN + FP) * (TN + FN)}} \qquad \textit{Equation 4-7}$$

4.2 Evaluation Results

After all implementations are complete, the evaluation process is performed using the criteria introduced in section 4.1. The assessment is accomplished utilizing the test dataset. Therefore, the models are exposed to samples they have not seen before and are demanded to classify them. The fault classes detected by the models are then compared against the ground truth for each data sample, and the evaluation metrics are calculated based on these two sets of values called *y_true* and *y_prediction*. It is noteworthy that all models go through two training phases except for the deep neural network. The tree-based algorithms, including the proposed Extra Tree classifier, experience a primary phase of training, after which SHAP feature importance is calculated. In the second phase, these algorithms are retrained using the top-15 features for each class to build the final models. Their results are compared to demonstrate how the retraining process helped improve accuracy and decrease training time. On the other hand, the deep neural network goes through a different training procedure in which feature importance is not calculated. This model is trained with all features ten times, and the results are aggregated by calculating mean values for the predictions. This section demonstrates how each instance differs from the others and proves that the deep learning implementation is robust and results are not randomly generated. Most importantly, the proposed method is compared against the baselines in this section to investigate its advantages and how it outperforms the baseline methods.

4.2.1 Training Process Results

This subsection is dedicated to the effects of the steps taken in the training process to enhance models' performance. Firstly, the developments throughout the feature importance calculations and models' retraining are investigated for the tree-based models. Thereafter, the deep learning performance is assessed within its numerous implementations.

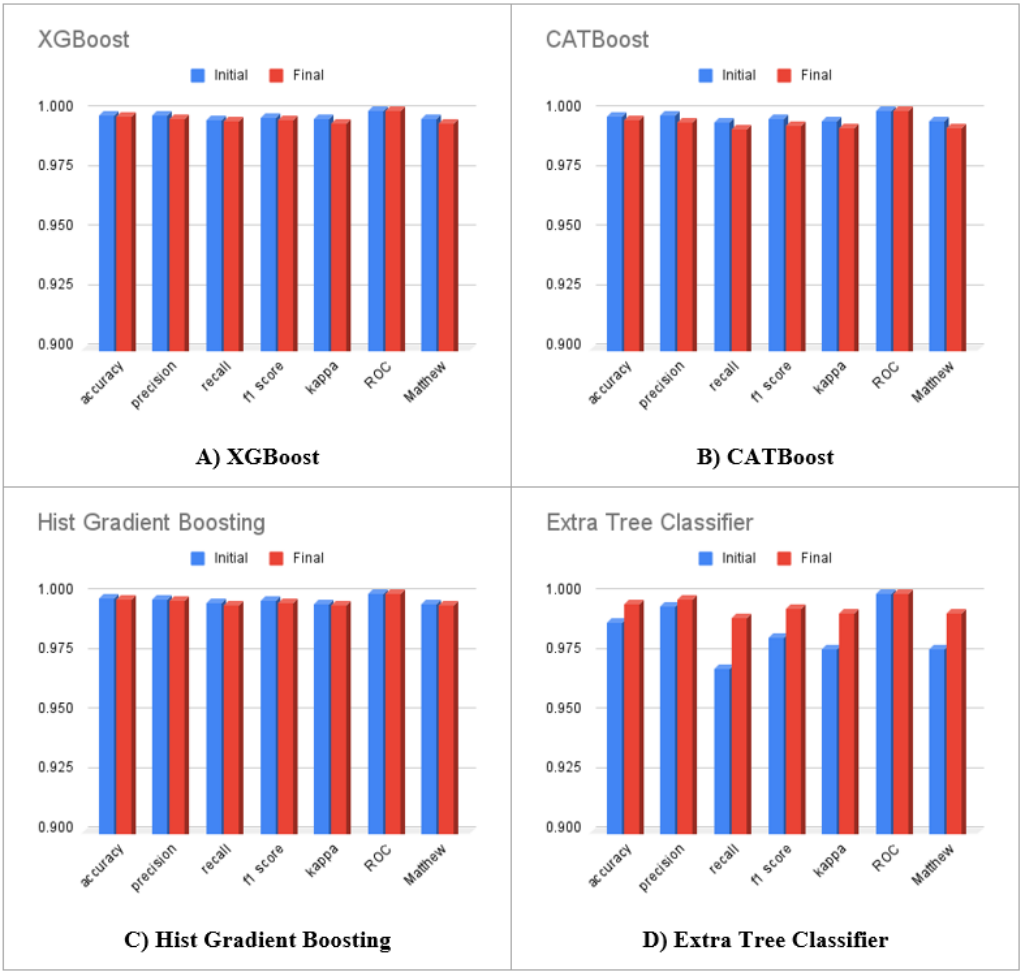


Figure 4-1 Comparison between the tree-based algorithms' performance before (initial) and after (final) the feature importance process.

4.2.1.1 Feature Importance Effects

As mentioned earlier, the tree-based models, including the proposed algorithm, were trained in two phases. Figure 4-1 demonstrates the effects of the second phase of the training on each model. In this figure, initial values represent the evaluation metrics before calculating feature importance, and the final values show models' performance after the second phase of learning with essential attributes extracted from the feature importance step. It is observed that most of these models did not change much while training with all features compared to training with the top-15 most influential features. However, while the rest of the algorithms experience a minor decline in terms of performance, the proposed method improves in all metrics. This proves that the proposed algorithm is a better choice for the problem at hand and is well-suited for this two-phase training procedure. Another significant matter is that the training times reduce by approximately half in most cases after the uninfluential features are omitted from the dataset, resulting in less training time and computational power required in monitoring devices. Figure 4-2 illustrates the training times for each algorithm with all features (initial) and with top-15 important features (final). Table 4-2 also summarizes all results in numerical values for additional proof.

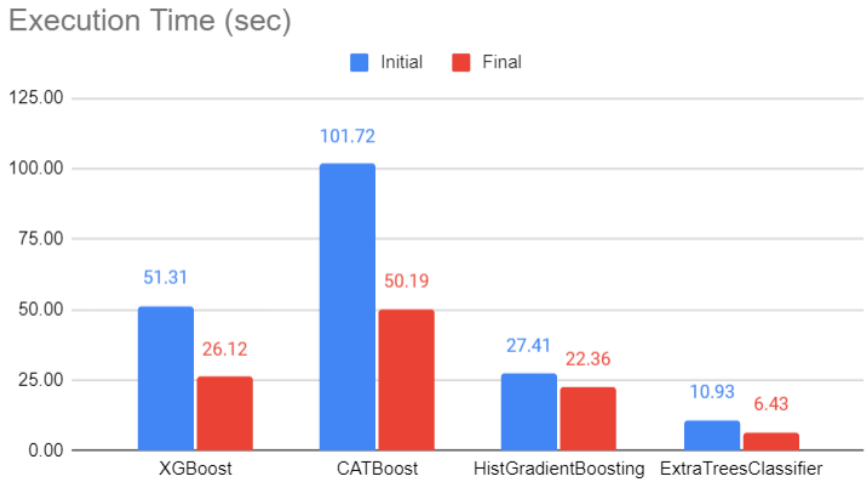


Figure 4-2 Comparison between the tree-based algorithms' execution time before (initial) and after (final) the feature importance process.

Table 4-2 Detailed overall comparison between the tree-based algorithms' performance before (initial) and after (final) the feature importance process.

	XGBoost		CATBoost		Hist Gradient Boosting		Extra Trees Classifier	
	Initial	**Final**	**Initial**	**Final**	**Initial**	**Final**	**Initial**	**Final**
Accuracy	0.9980	0.9972	0.9976	0.9960	0.9977	0.9973	0.9877	0.9955
Precision	0.9977	0.9965	0.9977	0.9948	0.9975	0.9967	0.9946	0.9973
Recall	0.9960	0.9953	0.9950	0.9923	0.9958	0.9951	0.9685	0.9897
F1 Score	0.9969	0.9959	0.9964	0.9935	0.9966	0.9959	0.9812	0.9935
Kappa	0.9962	0.9947	0.9954	0.9923	0.9956	0.9949	0.9764	0.9914
ROC	0.9999	0.9998	0.9999	0.9998	0.9998	0.9998	0.9998	0.9998
Matthew's	0.9962	0.9947	0.9954	0.9923	0.9956	0.9949	0.9766	0.9914
Ex. Time (sec)	51.31	26.12	101.72	50.19	27.41	22.36	10.93	6.43

4.2.1.2 Deep Learning Procedure

The deep neural network has also gone through multiple iterations of training. More specifically, the same architecture has been trained for ten separate iterations in which the initial weights are randomly set. Training sets and test sets have also been unsystematically sampled to maintain the randomness of iterations. After the model was trained in these steps, it was confronted with the test sets, and evaluation metrics were saved after each implementation. Figure 4-3 demonstrates the results of the five most essential metrics throughout this stage. This illustration proves that although the deep neural network was initialized randomly and received different training and test sets, it ended up with approximately the same results in terms of performance metrics. This process validates that the neural network was able to recognize the patterns related to each fault class, and its outcomes are not randomly generated. This proves that this model does not require the same preprocessing steps as the tree-based algorithms and can figure out the critical features.

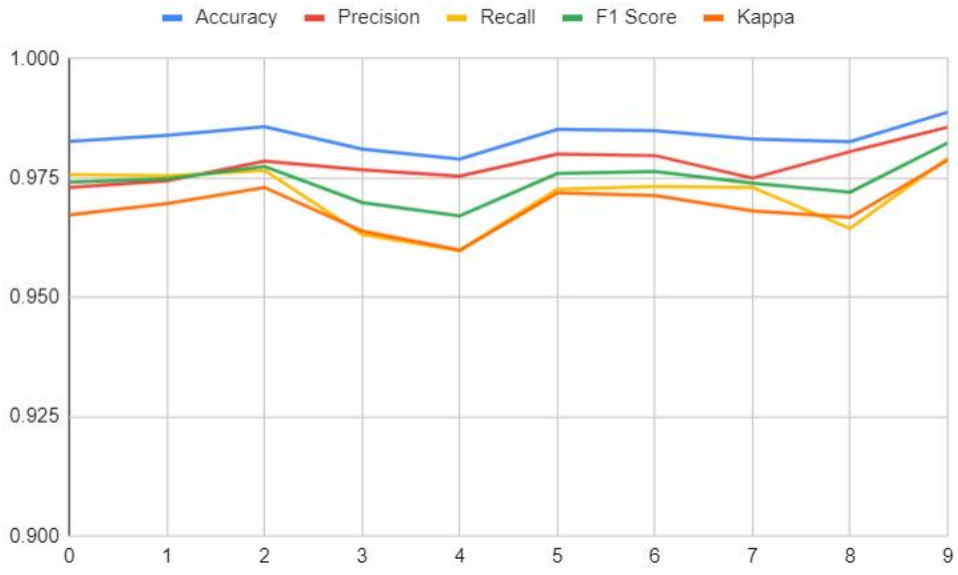
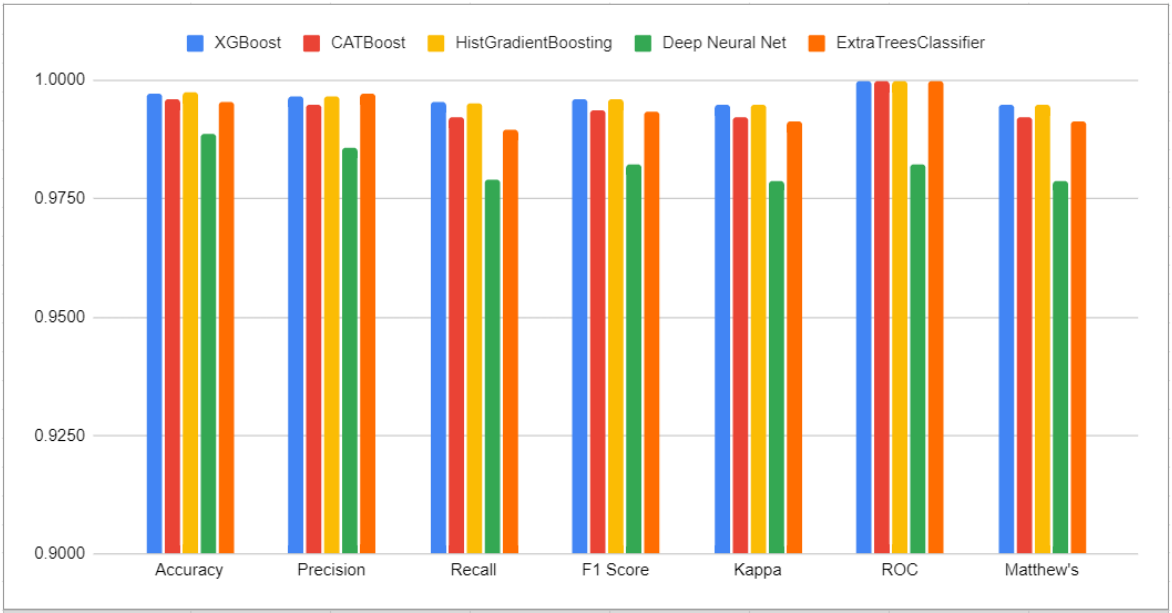


Figure 4-3 Evaluation results of the different training iterations of the deep neural network



	Accuracy	Precision	Recall	F1 Score	Kappa	ROC	Matthew's
XGBoost	0.9972	0.9965	0.9953	0.9959	0.9947	0.9998	0.9947
CATBoost	0.9960	0.9948	0.9923	0.9935	0.9923	0.9998	0.9923
HistGradientBoosting	0.9973	0.9967	0.9951	0.9959	0.9949	0.9998	0.9949
Deep Neural Net	0.9888	0.9856	0.9791	0.9824	0.9788	0.9823	0.9788
ExtraTreesClassifier	0.9955	0.9973	0.9897	0.9935	0.9914	0.9998	0.9914

Figure 4-4 A comparison between the implementations on the PHM challenge's dataset among all models and regarding all metrics.

4.2.2 Final Results

Figure 4-4 represents the comparison between the implementations on the PHM challenge's dataset between all models. The findings demonstrate that the proposed Extra Tree classifier performed exceptionally well on this classification task, with an accuracy of above 99 percent. Precision, recall, and F1 measures are also quite well in its case, demonstrating that the model is able to achieve a high percentage of total relevant outcomes precisely classified. The closeness of the Kappa value to one also indicates a strong correlation between the actual and predicted classes categorized by this model. Despite the similar results between the proposed models

and baseline algorithms, the proposed method performs slightly better in terms of precision and comparable outcomes in other metrics.

However, the main advantage of the proposed algorithm is its short training time and limited resources required for training. As figure 4-5 illustrates, the proposed model is trained in less than 7 seconds with the enhanced dataset containing the top-15 important features. The next best result would be dedicated to the Hist Gradient Boosting approach, which is 3.5 times longer than the proposed method, revealing that the Extra Tree algorithm provides comparable metrics in much less time.

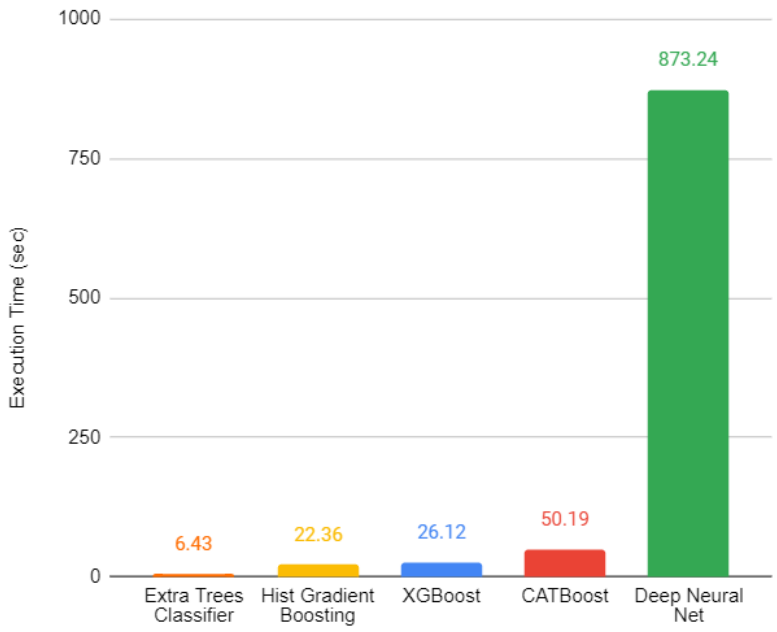


Figure 4-5 Training time comparison between the implementations (seconds)

4.2.3 Root Cause Analysis

Root cause analysis is a technique for identifying and analyzing the reasons for issues, deviations, and failures. This strategy identifies the root cause of a process

problem and groups issues with the same cause together. The report then delves into the elements that are producing issues. To accomplish root cause analysis, ML algorithms cluster the issues. The algorithms then analyze the causes affecting these difficulties in order to establish which elements are linked to which issues and which may be the root of the problem. These algorithms scan the data for apparent patterns and relationships.

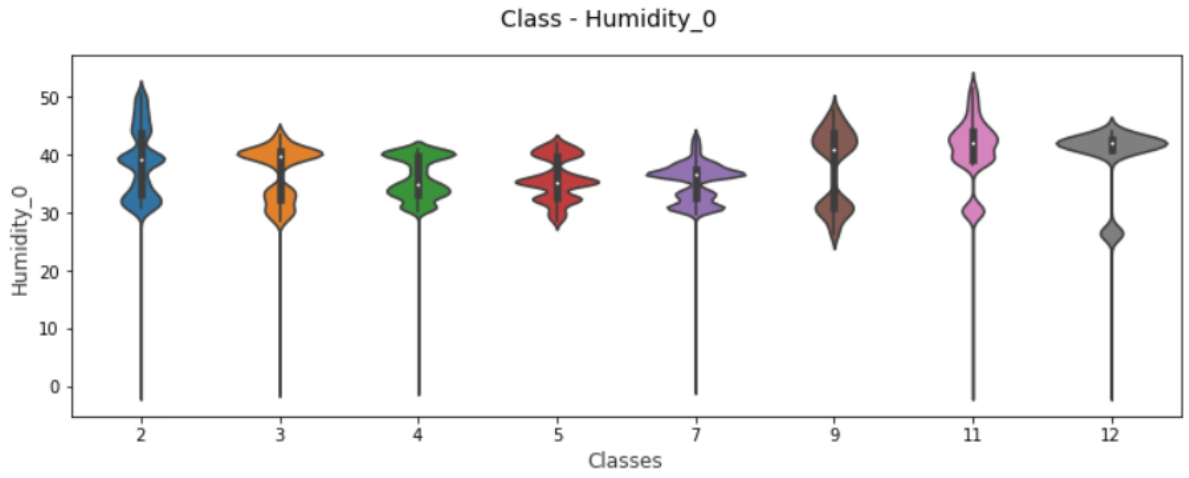


Figure 4-6 Violin plot of the effects of humidity on fault classes on the PHM dataset.

The root causes are investigated based on the features inducing errors in each fault category in this research. The effects of each attribute in performing classifications were calculated in the feature importance phase, as discussed in section 3.5, and the most influential elements were diagnosed. The process of analyzing the root causes was performed in a class-based procedure. It is noteworthy that the analysis was conducted on all features, not on the top 15 selected after the feature importance stage. Visualizations demonstrate that defects are provoked by a limited number of flaws in each category, and multiple common issues appear in more than one fault. For instance, figure 4-6 shows a violin plot of how humidity affects all fault classes and can cause difficulties in mechanical environments. Temperature and Fuse Cycle

Durations are among the other common problems in the PHM dataset's faulty cases. However, different values of a single cause can influence different classes. For example, figure 4-7 displays the distribution of CPU temperature values in cases where it caused errors in class 3 versus the cases where no fault occurred in class 0. Yet, CPU temperature was an effective attribute in both these cases.

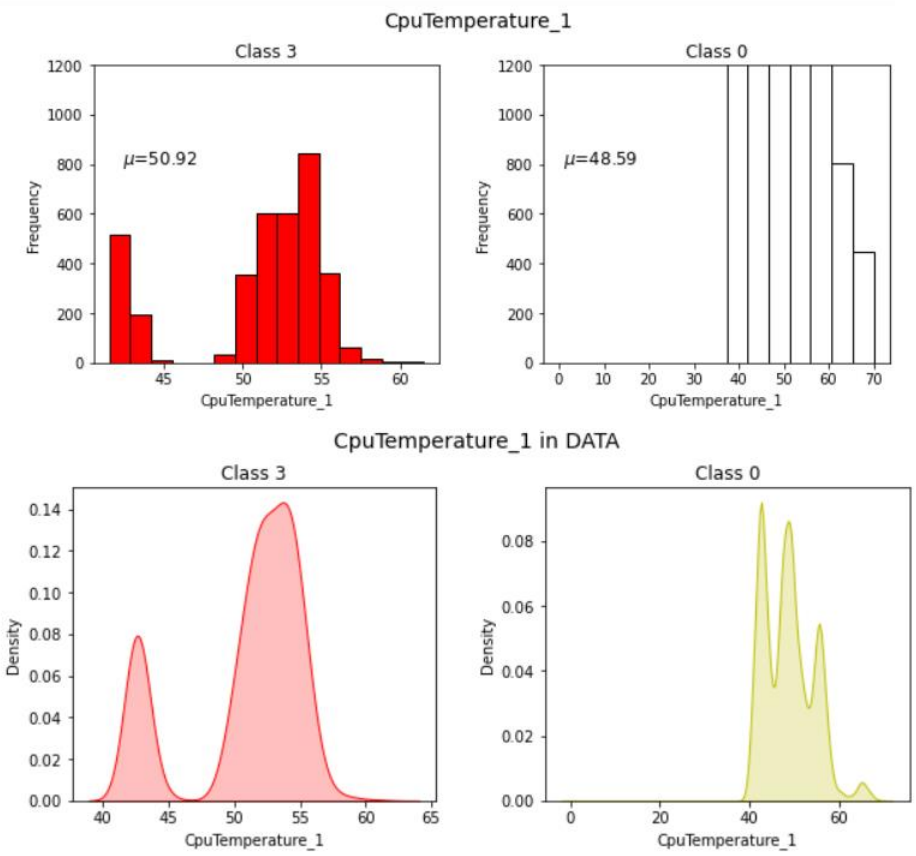


Figure 4-7 Comparison between cases where CPU temperature caused a fault (class 3) and value distributions where it caused no fault (class 0)

On the other hand, causes such as pressure are only seen in a particular class. Figure 4-8 demonstrates how pressure affects only the third class and has little or no effect on other cases. Ultimately, figure 4-9 concludes this part by introducing the top-5 features affecting the faulty classes the most. It is worth mentioning that common

features such as humidity and temperature are omitted from all classes except the ones they have the most effects on. Therefore, only unique features are taken into consideration for each fault class.

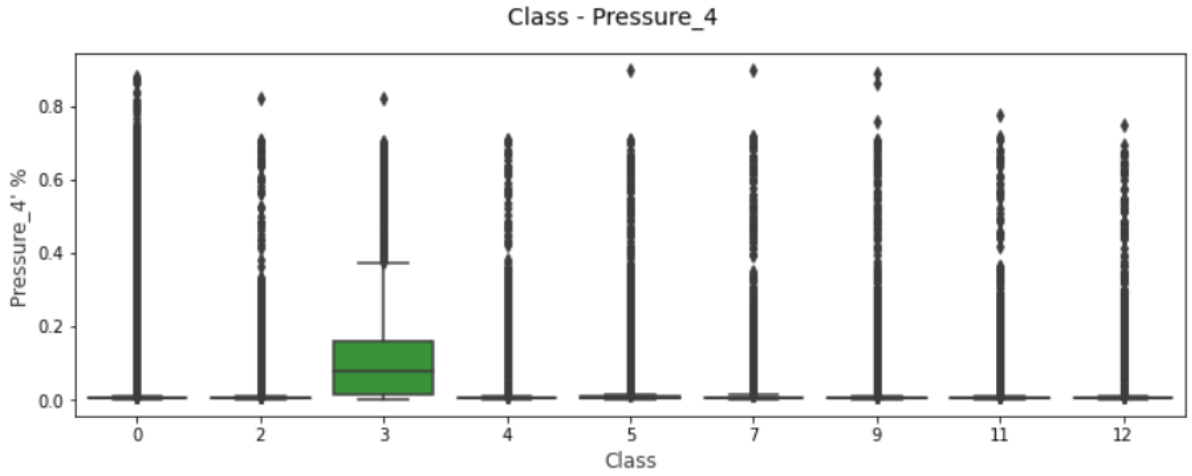


Figure 4-8 Effects of pressure on the third fault class compared to little or no effects on the other classes.

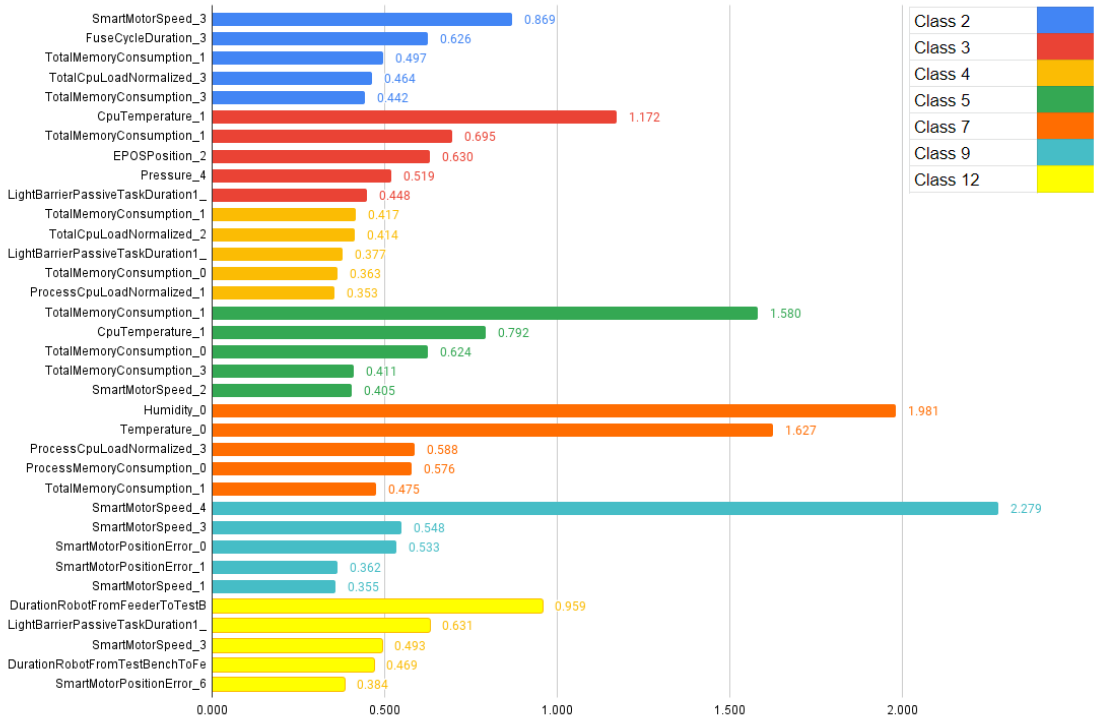


Figure 4-9 Top-5 most influential features in faulty classes based on the SHAP feature importance.

4.2.4 Fault Diagnosis

The primary goal of this thesis was to identify faults intelligently. Now that the robustness of the proposed algorithm is verified, a fault diagnosis can be performed using this method. These results are based on the root cause analysis and the feature importance values calculated for each fault class. In other words, the essential signals in the feature importance analysis indicate the fundamental cause of an issue. More studies into the data are carried out to better understand the origins of the problems and their physical interpretation, particularly on the most significant signals. Table 4-3 summarizes these most critical signals for each class. This table displays the signals causing most cases in the samples available in each class. Although some

signals might have large importance values, they do not trigger faults in a majority of circumstances. For instance, common signals such as humidity cannot be the root cause of any classes since they are available in all categories. The only case where such issues are influential is class 7, in which humidity and temperature are among the top difficulties causing faults.

Table 4-3 Most influential signals for each class.

Class	Most Important Signals	Fault Interpretation
2	FuseCycleDuration_3 FeederAction2_0	the fuses have less cycle duration in this class
3	CpuTemperature_1 Pressure_4	Mean CPU temperature of the control and data acquisition computer is above average.
4	TotalCpuLoadNormalized_2	Higher Derivative-based trend for CPU
5	TotalMemoryConsumption_0 ProcessMemoryConsumption_0	Fluctuation in number of data acquisition for Memory consumption.
7	TotalMemoryConsumption_1 Humidity_0 Temperature_0	Difference in environmental condition and memory consumption acquired value.
9	SmartMotorSpeed_0 SmartMotorPositionError_0	Reduction in number of data acquired for the speed of the motor.
11	SmartMotorSpeed_3 SmartMotorPositionError_3 DurationRobotFromFeederToTestBench_6	Rise in the motor speed value; as a consequence, less duration to move from the feeder to the test bench.
12	DurationRobotFromFeederToTestBench_6	Higher Frequency of the measurements for duration to move

4.3 Discussion

It is observed that the proposed method outperforms all baseline algorithms regarding the computational complexity and training time and, therefore, is desirable in the PHM's dataset. It also reveals the best precision among all the

implementations, and the rest of the metrics are quite comparable in this case. Although these measurements are pretty close to each other, achieving such results is not a trivial task, and prior experiments proved that several other algorithms are not able to perform desirably on this task. Figure 4-6 shows how five famous machine learning algorithms perform on the same dataset using the same steps on the PHM fault classification dataset. It is noticed that none of these algorithms, including the Linear Discriminant Analysis (LDA), Logistic Regression, K-Nearest Neighbors (KNN), AdaBoost, and the Support Vector Machine (SVM), were able to accomplish an accuracy higher than 95%. This is proof that choosing tree-based models and deep neural networks was the right preference since they all correctly classified more than 99% of the test samples.

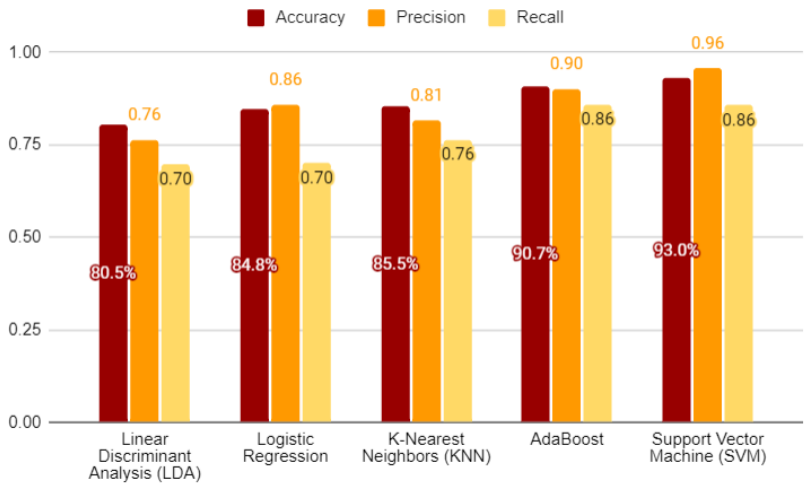


Figure 4-10 Test results on five popular machine learning approaches for the PHM dataset.

The confusion matrix from the test samples classified by the proposed model also demonstrates the robustness of this algorithm, as seen in Figure 4-7. The primary diagonal in this matrix indicates the correct classifications, in which the predicted class is equal to the actual one. The matrix reveals that the model can classify faults

correctly for all cases except for 180 errors out of over 16 thousand test samples, confirming that the classification task is performed almost flawlessly. Ultimately, the final root cause analysis establishes that the proposed model does not act as a black box. Its results are interpretable and enable researchers to understand the causes and effects of each attribute on each fault in the dataset.

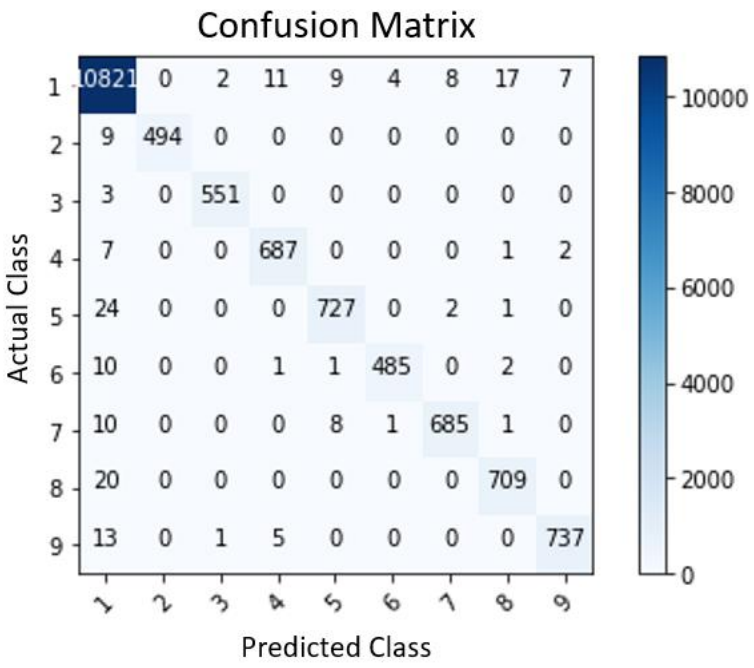


Figure 4-11 The confusion matrix resulted from the Extra Tree classification on the test set. The primary diagonal shows the correct classification of each defect.

Chapter 5

5 Conclusion and Recommendations

5.1 Conclusion

This research proposed an Extra Tree classifier to detect mechanical faults. The study is performed in multiple stages, in which different preprocessing steps and machine learning algorithms were examined to perform the classification task at hand. Experiments revealed that tree-based algorithms and deep neural networks could achieve the best results among all possible methods. Therefore, four algorithms based on the concepts of trees were picked, namely, the XGBoost, CATBoost, Hist Gradient Boosting, and the Extra Tree classifier. These methods went through several stages of data preprocessing, in which the dataset was cleaned, and the important features in fulfilling the desirable classification were revealed using the SHAP (SHapley Additive exPlanations) approach. A fully-connected deep learning algorithm was also designed to perform the same task without the preprocessing phases. Such algorithms are developed to identify essential features by themselves and do not require ordinary data processing methods.

Moreover, an evaluation process was designed to assess models' performance regarding various metrics. Investigations confirmed that the five chosen architectures could achieve accuracies larger than 99%, and their results are

comparable in most cases. The Extra Tree classifier is chosen as the proposed algorithm since it achieves the same results in much less time. Although there are minor differences regarding the evaluation metrics, the most important superiority of the proposed algorithm is its short training time and capacity to work with limited computational power. While accuracy and similar metrics are critical to showing the model's ability to detect faults, performing this task with fewer computations and in a swifter way is also desirable in mechanical industries where computer hardware is not updated regularly. While the Extra Tree classifier is trained and ready to use in less than 7 seconds, the deep learning model took approximately 870 seconds to complete the same process. Although the other tree-based baseline algorithms are quicker than the DL architecture, they were also unable to outperform the proposed method. As a result, it is confirmed that the proposed algorithm is both efficient in terms of computational power and robust regarding classification quality. At the end of this stage, a root cause analysis is conducted to explore the causality of each fault, revealing many of these defects are provoked by a limited number of issues such as humidity and temperature. This step proves that the proposed model enables the users to diagnose the causes of each fault occurring and does not act as a black box, which is another advantage compared to deep neural networks.

5.2 Recommendations

There are numerous paths that can be taken in order to continue this research. As section 2 demonstrated, the future of the Intelligent Fault Diagnosis research is established on a path where more data can be utilized. The largeness of datasets can assist with using modern deep learning architectures, and researchers have shown that these models can improve significantly by the growth of their training data.

Therefore, it is efficient to use generative models to enlarge the datasets and improve the amount of data the models receive. On the other hand, many research works have confirmed that using models' experience with former problems can improve their efficiency on similar tasks. As a result, the larger datasets generated by the generative models can be coupled with transfer learning approaches to take advantage of robust models' experiences for the IFD process. Using pre-trained models can also decrease the training time and amount of computational power required for ML-based models.

On the other hand, unsupervised learning methods can be associated with the supervised approaches to improve the cause analysis and the configurations resulting in each fault. More specifically, clustering algorithms can be utilized to analyze data samples within each class and group them by the various cases of configurations that can cause a particular fault. This method is helpful in completing the analysis of root causes and will help configure faulty cases ideally. Moreover, although the Bayesian optimization algorithm was employed to improve the model's hyperparameters, more optimization can be performed using more recent algorithms and including more extensive parameter settings.

Ultimately, it is known that taking advantage of such algorithms can be complicated and non-practical for mechanical engineers without any functional background in machine learning engineering. Since maintaining ML-based algorithms and updating them using new data acquired in the mechanical processes can be challenging for non-experts, using Automated Machine Learning (AutoML) approaches can be practical in this field. The process of automating the activities of applying machine learning to real-world situations is known as AutoML. Every stage of a machine learning process, from launching with a raw dataset to developing a

machine learning model suitable for deployment, could be covered in AutoML. Therefore, it is recommended to use such procedures in order to facilitate the machine learning process and decrease the challenges for mechanical experts.

Bibliography

[1] S. Shao, S. Member, S. Mcaleer, R. Yan, and S. Member, "Highly-Accurate Machine Fault Diagnosis Using Deep Transfer Learning," *IEEE Trans. Ind. Informatics*, vol. PP, no. c, p. 1, 2018, doi: 10.1109/TII.2018.2864759.

[2] T. Van Tung and B. Yang, "Machine Fault Diagnosis and Prognosis : The State of The Art," vol. 2, no. 1, pp. 61–71, 2009.

[3] X. Guo, L. Chen, and C. Shen, "Hierarchical adaptive deep convolution neural network and its application to bearing fault diagnosis," *Measurement*, 2016, doi: 10.1016/j.measurement.2016.07.054.

[4] M. Cerrada, G. Zurita, D. Cabrera, R. V. Sánchez, M. Artés, and C. Li, "Fault diagnosis in spur gears based on genetic algorithm and random forest," *Mech. Syst. Signal Process.*, vol. 70–71, pp. 87–103, 2016, doi: 10.1016/j.ymssp.2015.08.030.

[5] B. Cai, L. Huang, and M. Xie, "Bayesian Networks in Fault Diagnosis," vol. 3203, no. c, 2017, doi: 10.1109/TII.2017.2695583.

[6] M. Gan, C. Wang, and C. Zhu, "Construction of hierarchical diagnosis network based on deep learning and its application in the fault pattern recognition of rolling element bearings," *Mech. Syst. Signal Process.*, pp. 1–13, 2015, doi: 10.1016/j.ymssp.2015.11.014.

[7] W. Lu *et al.*, "Deep Model Based Domain Adaptation for Fault Diagnosis," vol. 0046, no. c, 2016, doi: 10.1109/TIE.2016.2627020.

[8] L. Jing, M. Zhao, P. Li, and X. Xu, "A convolutional neural network based

feature learning and fault diagnosis method for the condition monitoring of gearbox," *Measurement*, 2017, doi: 10.1016/j.measurement.2017.07.017.

[9] C. Cheng, J. Wang, H. Chen, Z. Chen, H. Luo, and P. Xie, "A review of intelligent fault diagnosis for high-speed trains: Qualitative approaches," *Entropy*, vol. 23, no. 1, pp. 1–33, 2021, doi: 10.3390/e23010001.

[10] C. Kara Mostefa Khelil, B. Amrouche, A. soufiane Benyoucef, K. Kara, and A. Chouder, "New Intelligent Fault Diagnosis (IFD) approach for grid-connected photovoltaic systems," *Energy*, vol. 211, pp. 1–18, 2020, doi: 10.1016/j.energy.2020.118591.

[11] W. A. Smith and R. B. Randall, "Rolling element bearing diagnostics using the Case Western Reserve University data : A benchmark study," *Mech. Syst. Signal Process.*, pp. 1–32, 2015, doi: 10.1016/j.ymssp.2015.04.021.

[12] H. Jiang, F. Wang, H. Shao, and H. Zhang, "Rolling bearing fault identification using multilayer deep learning convolutional neural network," pp. 138–149, 2017, doi: 10.21595/jve.2016.16939.

[13] M. Marrone, "Application of entity linking to identify research fronts and trends," *Scientometrics*, vol. 122, no. 1, pp. 357–379, 2020, doi: 10.1007/s11192-019-03274-x.

[14] A. Bhandare, M. Bhide, P. Gokhale, and R. Chandavarkar, "Applications of Convolutional Neural Networks," vol. 7, no. 5, pp. 2206–2215, 2016.

[15] J. J. Premkumar, "Machine Learning in Automatic Speech Recognition: A Survey," *IETE Tech. Rev.*, pp. 240–251, 2015.

[16] S. Pouyanfar, M. Shyu, S. Chen, and S. S. Iyengar, "A Survey on Deep Learning : Algorithms , Techniques, and Applications," *ACM Comput. Surv.*,

vol. 51, no. 5, 2018.

[17] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, “Deep learning,” *Nature*, vol. 1, no. 2, 2016.

[18] J. Schmidhuber, “Deep Learning in neural networks: An overview,” *Neural Networks*, vol. 61, pp. 85–117, 2015, doi: 10.1016/j.neunet.2014.09.003.

[19] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, 2017, doi: 10.2307/j.ctt1d98bxx.10.

[20] R. DiPietro and G. D. Hager, *Deep learning: RNNs and LSTM*. Elsevier Inc., 2019.

[21] F. Rafique, L. Fu, and R. Mai, “End to end machine learning for fault detection and classification in power transmission lines,” *Electr. Power Syst. Res.*, vol. 199, no. June, p. 107430, 2021, doi: 10.1016/j.epsr.2021.107430.

[22] J. Jiao, M. Zhao, J. Lin, and K. Liang, “A comprehensive review on convolutional neural network in machine fault diagnosis,” *Neurocomputing*, vol. 417, pp. 36–63, 2020, doi: 10.1016/j.neucom.2020.07.088.

[23] L. Guo, Y. Lei, S. Xing, T. Yan, and N. Li, “Deep Convolutional Transfer Learning Network : A New Method for Intelligent Fault Diagnosis of Machines With Unlabeled Data,” vol. 66, no. 9, pp. 7316–7325, 2019.

[24] J. Zou, Y. Han, and S. So, “Overview of Artificial Neural Networks,” 2008.

[25] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” *Comput. Vis. Pattern Recognit.*, pp. 770–778, 2016.

[26] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for

Large-Scale Image Recognition," pp. 1–14, 2015.

[27] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-Decem, pp. 2818–2826, 2016, doi: 10.1109/CVPR.2016.308.

[28] X. Pang, Y. Zhou, P. Wang, W. Lin, and V. Chang, "An innovative neural network approach for stock market prediction," *J. Supercomput.*, vol. 76, no. 3, pp. 2098–2118, 2020, doi: 10.1007/s11227-017-2228-y.

[29] L. Zhang, J. Lin, B. Liu, Z. Zhang, X. Yan, and M. Wei, "A Review on Deep Learning Applications in Prognostics and Health Management," *IEEE Access*, vol. 7, pp. 162415–162438, 2019, doi: 10.1109/ACCESS.2019.2950985.

[30] I. Goodfellow, "Generative Modeling Generative Modeling," 2016.

[31] M. Kang and N. J. Jameson, "Machine Learning: Fundamentals," *Progn. Heal. Manag. Electron.*, pp. 85–109, 2018, doi: 10.1002/9781119515326.ch4.

[32] F. Pedregosa *et al.*, "Scikit-learn: Machine Learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.

[33] Y. Bengio, "Learning Deep Architectures for AI," *Found. Trends® Mach. Learn.*, vol. 2, no. 1, pp. 1–127, 2009, [Online]. Available: http://www.nowpublishers.com/product.aspx?product=MAL&doi=2200000006.

[34] S. Ioffe and C. Szegedy, "Batch Normalization : Accelerating Deep Network Training by Reducing Internal Covariate Shift," *Icml. (2015)*, 2015.

[35] H. Winner, S. Hakuli, F. Lotz, and C. Singer, *Fundamentals of Machine Vision*. 2015.

[36] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfittin," *J. Mach. Learn. Res.*, vol. 15, pp. 1929–1958, 2014, doi: 10.1109/CVPR.2018.00797.

[37] G. Dorgo, A. Palazoglu, and J. Abonyi, "Decision trees for informative process alarm definition and alarm-based fault classification," *Process Saf. Environ. Prot.*, vol. 149, pp. 312–324, 2021, doi: 10.1016/j.psep.2020.10.024.

[38] P. Tamilselvan and P. Wang, "Failure diagnosis using deep belief learning based health state classification," *Reliab. Eng. Syst. Saf.*, vol. 115, pp. 124–135, 2013, doi: 10.1016/j.ress.2013.02.022.

[39] W. Zhang, G. Peng, C. Li, Y. Chen, and Z. Zhang, "A New Deep Learning Model for Fault Diagnosis with Good Anti-Noise and Domain Adaptation Ability on Raw Vibration Signals," 2017, doi: 10.3390/s17020425.

[40] R. Liu, B. Yang, E. Zio, and X. Chen, "Artificial intelligence for fault diagnosis of rotating machinery : A review," *Mech. Syst. Signal Process.*, vol. 108, pp. 33–47, 2018, doi: 10.1016/j.ymssp.2018.02.016.

[41] R. Miotto, F. Wang, S. Wang, X. Jiang, and J. T. Dudley, "Deep learning for healthcare: Review, opportunities and challenges," *Brief. Bioinform.*, vol. 19, no. 6, pp. 1236–1246, 2017, doi: 10.1093/bib/bbx044.

[42] R. K. Patel and V. K. Giri, "Feature selection and classification of mechanical fault of an induction motor using random forest classifier,"

Perspect. Sci., vol. 8, pp. 334–337, 2016, doi: 10.1016/j.pisc.2016.04.068.

[43] Y. Lei, F. Jia, J. Lin, S. Xing, and S. X. Ding, "An Intelligent Fault Diagnosis Method Using Unsupervised Feature Learning Towards Mechanical Big Data," vol. 0046, no. c, 2016, doi: 10.1109/TIE.2016.2519325.

[44] Z. Wang, G. Li, L. Yao, X. Qi, and J. Zhang, "Data-driven fault diagnosis for wind turbines using modified multiscale fluctuation dispersion entropy and cosine pairwise-constrained supervised manifold mapping," *Knowledge-Based Syst.*, vol. 228, p. 107276, 2021, doi: 10.1016/j.knosys.2021.107276.

[45] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 10, pp. 1345–1359, 2010, doi: 10.1109/TKDE.2009.191.

[46] A. Bernieri, M. D'Apuzzo, L. Sansone, and M. Savastano, "A Neural Network Approach for Identification and Fault Diagnosis on Dynamic Systems," vol. 43, no. 6, 1994.

[47] J. Han, M. Kamber, and J. Pei, "Data Mining: Concepts and Techniques," *Data Min. Concepts Tech.*, 2012, doi: 10.1016/C2009-0-61819-5.

[48] S. Shai Shalev and D. Shai Ben, *Understanding Machine Learning*. 2014.

[49] Z. Gao, S. Member, C. Cecati, F. Ieee, and S. X. Ding, "A Survey of Fault Diagnosis and Fault - Tolerant Techniques Part II : Fault Diagnosis with Knowledge - Based and Hybrid / Active Approaches," vol. 0046, no. c, 2015, doi: 10.1109/TIE.2015.2419013.

[50] B. Samanta, "Gear fault detection using artificial neural networks and support vector machines with genetic algorithms," vol. 18, pp. 625–644, 2004, doi: 10.1016/S0888-3270(03)00020-7.

[51] V. Ã. Sugumaran, V. Muralidharan, and K. I. Ramachandran, “Feature selection using Decision Tree and classification through Proximal Support Vector Machine for fault diagnostics of roller bearing,” vol. 21, pp. 930–942, 2007, doi: 10.1016/j.ymssp.2006.05.004.

[52] X. Zhang, Y. Liang, and J. Zhou, “A novel bearing fault diagnosis model integrated permutation entropy , ensemble empirical mode decomposition and optimized SVM,” *MEASUREMENT*, vol. 69, pp. 164–179, 2015, doi: 10.1016/j.measurement.2015.03.017.

[53] S. Datta and S. Sarkar, “A review on different pipeline fault detection methods,” *J. Loss Prev. Process Ind.*, vol. 41, pp. 97–106, 2016, doi: 10.1016/j.jlp.2016.03.010.

[54] A. Stetco *et al.*, “Machine Learning Methods for Wind Turbine Condition Monitoring: A Review,” *Renew. Energy*, 2018, doi: 10.1016/j.renene.2018.10.047.

[55] P. Akulwar, S. Pardeshi, and A. Kamble, “Survey on different data mining techniques for prediction,” *Proc. Int. Conf. I-SMAC (IoT Soc. Mobile, Anal. Cloud), I-SMAC 2018*, pp. 513–519, 2019, doi: 10.1109/I-SMAC.2018.8653734.

[56] R. Yan, Z. Ma, Y. Zhao, and G. Kokogiannakis, “A decision tree based data-driven diagnostic strategy for air handling units,” *Energy Build.*, vol. 133, pp. 37–45, 2016, doi: 10.1016/j.enbuild.2016.09.039.

[57] Y. Zhao, L. Yang, and B. Lehman, “Decision Tree-Based Fault Detection and Classification in Solar Photovoltaic Arrays,” pp. 93–99, 2012.

[58] S. J. Norvig and P. Russell, *Artificial Iintelligence: Modern Approach*, no. 2.

1995.

[59] J. Zhang *et al.*, “XGBoost Classifier for Fault Identification in Low Voltage Neutral Point Ungrounded System,” pp. 1767–1771, 2019.

[60] Z. Wu, M. Zhou, Z. Lin, X. Chen, and Y. Huang, “Improved Genetic Algorithm and XGBoost Classifier for Power Transformer Fault Diagnosis,” vol. 9, no. October, pp. 1–10, 2021, doi: 10.3389/fenrg.2021.745744.

[61] S. Behrouzi, Z. Shafaeipour Sarmoor, K. Hajsadeghi, and K. Kavousi, “Predicting scientific research trends based on link prediction in keyword networks,” *J. Informetr.*, vol. 14, no. 4, p. 101079, 2020, doi: 10.1016/j.joi.2020.101079.

[62] Z. Xu and D. Yu, “A Bibliometrics analysis on big data research (2009–2018),” *J. Data, Inf. Manag.*, vol. 1, no. 1–2, pp. 3–15, 2019, doi: 10.1007/s42488-019-00001-2.

[63] L. Puggini, J. Doyle, and S. McLoone, “Fault Detection using Random Forest Similarity Distance,” *IFAC-PapersOnline*, pp. 583–588, 2015, doi: 10.1016/j.ifacol.2015.09.589.

[64] M. Ashok Mahant and V. Pellakuri, “Innovative supervised machine learning techniques for classification of data,” *Mater. Today Proc.*, no. xxxx, 2021, doi: 10.1016/j.matpr.2020.11.092.

[65] J. Tian, C. Morillo, M. H. Azarian, and M. Pecht, “Kurtosis-Based Feature Extraction Coupled With K -Nearest Neighbor Distance Analysis,” vol. 63, no. 3, pp. 1793–1803, 2016.

[66] S. Naik and E. Koley, “Fault Detection and Classification scheme using KNN for AC / HVDC Transmission Lines,” no. Icces, pp. 1–5, 2019.

[67] S. Ben-david, *Understanding Machine Learning : From Theory to Algorithms*. Cambridge University Press, 2014.

[68] V. Muralidharan and V. Sugumaran, “A comparative study of Naïve Bayes classifier and Bayes net classifier for fault diagnosis of monoblock centrifugal pump using wavelet analysis,” *Appl. Soft Comput. J.*, vol. 12, no. 8, pp. 2023–2029, 2012, doi: 10.1016/j.asoc.2012.03.021.

[69] Y. Zhao, J. Wen, F. Xiao, X. Yang, and S. Wang, “Diagnostic Bayesian networks for diagnosing air handling units faults – part I: Faults in dampers, fans, filters and sensors,” *Appl. Therm. Eng.*, vol. 111, pp. 1272–1286, 2017, doi: 10.1016/j.applthermaleng.2015.09.121.

[70] Y. Lecun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015, doi: 10.1038/nature14539.

[71] N. Ketkar, *Deep Learning with Python*. 2017.

[72] C. Lu, Z. Wang, W. Qin, and J. Ma, “Fault diagnosis of rotary machinery components using a stacked denoising autoencoder-based health state identi fi cation,” *Signal Processing*, vol. 130, pp. 377–388, 2017, doi: 10.1016/j.sigpro.2016.07.028.

[73] F. Xia, W. Wang, T. M. Bekele, and H. Liu, “Big Scholarly Data: A Survey,” *IEEE Trans. Big Data*, vol. 3, no. 1, pp. 18–35, 2017, doi: 10.1109/tbdata.2016.2641460.

[74] H. Liu, J. Zhou, Y. Zheng, W. Jiang, and Y. Zhang, “Fault diagnosis of rolling bearings with recurrent neural network- based autoencoders,” *ISA Trans.*, pp. 1–12, 2018, doi: 10.1016/j.isatra.2018.04.005.

[75] Z. Chen, C. Li, and R. Sanchez, “Gearbox Fault Identification and

Classification with Convolutional Neural Networks," vol. 2015, 2015.

[76] R. Zhao, R. Yan, Z. Chen, K. Mao, P. Wang, and R. X. Gao, "Deep learning and its applications to machine health monitoring," *Mech. Syst. Signal Process.*, vol. 115, pp. 213–237, 2019, doi: 10.1016/j.ymssp.2018.05.050.

[77] G. Hu, T. Zhou, and Q. Liu, "Data-Driven Machine Learning for Fault Detection and Diagnosis in Nuclear Power Plants: A Review," *Front. Energy Res.*, vol. 9, no. May, pp. 1–12, 2021, doi: 10.3389/fenrg.2021.663296.

[78] F. Jia, Y. Lei, J. Lin, X. Zhou, and N. Lu, "Deep neural networks : A promising tool for fault characteristic mining and intelligent diagnosis of rotating machinery with massive data," 2015, doi: 10.1016/j.ymssp.2015.10.025.

[79] W. Sun, S. Shao, R. Zhao, R. Yan, X. Zhang, and X. Chen, "A Sparse Auto-encoder-Based Deep Neural Network Approach for Induction Motor Faults Classification," *MEASUREMENT*, 2016, doi: 10.1016/j.measurement.2016.04.007.

[80] E. Chong, C. Han, and F. C. Park, "Deep learning networks for stock market analysis and prediction: Methodology, data representations, and case studies," *Expert Syst. Appl.*, vol. 83, pp. 187–205, 2017, doi: 10.1016/j.eswa.2017.04.030.

[81] G. Liu, H. Bao, and B. Han, "A Stacked Autoencoder-Based Deep Neural Network for Achieving Gearbox Fault Diagnosis," *Math. Probl. Eng.*, vol. 2018, 2018, doi: 10.1155/2018/5105709.

[82] Z. Chen and W. L. Member, "Multisensor Feature Fusion for Bearing Fault Diagnosis Using Sparse Autoencoder and Deep Belief Network," pp. 1–10,

2017.

[83] B. A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” 2017.

[84] L. He, G. Wang, and Z. Hu, “Learning depth from single images with deep neural network embedding focal length,” *IEEE Trans. Image Process.*, vol. 27, no. 9, pp. 4676–4689, 2018, doi: 10.1109/TIP.2018.2832296.

[85] L. Wen, X. Li, L. Gao, and Y. Zhang, “A New Convolutional Neural Network Based Data-Driven Fault Diagnosis Method,” vol. 0046, no. c, 2017, doi: 10.1109/TIE.2017.2774777.

[86] C. Lu, Z. Wang, and B. Zhou, “Advanced Engineering Informatics Intelligent fault diagnosis of rolling bearing using hierarchical convolutional network based health state classification,” *Adv. Eng. Informatics*, vol. 32, pp. 139–151, 2017, doi: 10.1016/j.aei.2017.02.005.

[87] O. Janssens *et al.*, “Convolutional Neural Network Based Fault Detection for Rotating Machinery,” 2016, doi: 10.1016/j.jsv.2016.05.027.

[88] M. Xia *et al.*, “Fault Diagnosis for Rotating Machinery Using Multiple Sensors and Convolutional Neural,” vol. 4435, no. c, pp. 1–9, 2017, doi: 10.1109/TMECH.2017.2728371.

[89] W. Zhang, C. Li, G. Peng, Y. Chen, and Z. Zhang, “A deep convolutional neural network with new training methods for bearing fault diagnosis under noisy environment and different working load,” *Mech. Syst. Signal Process.*, vol. 100, pp. 439–453, 2018, doi: 10.1016/j.ymssp.2017.06.022.

[90] X. Li, Q. Ding, and J. Sun, “Remaining Useful Life Estimation in Prognostics Using Deep Convolution Neural Networks,” *Reliab. Eng. Syst.*

Saf., 2017, doi: 10.1016/j.ress.2017.11.021.

[91] S. Wang, J. Xiang, Y. Zhong, and Y. Zhou, "Convolutional neural network-based hidden Markov models for rolling element bearing fault identification," *Knowledge-Based Syst.*, vol. 0, pp. 1–12, 2017, doi: 10.1016/j.knosys.2017.12.027.

[92] W. Byeon, T. M. Breuel, F. Raue, and M. Liwicki, "Scene labeling with LSTM recurrent neural networks," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 07-12-June, pp. 3547–3555, 2015, doi: 10.1109/CVPR.2015.7298977.

[93] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Comput.*, vol. 9, pp. 1735–1780, 1997, doi: 10.17582/journal.pjz/2018.50.6.2199.2207.

[94] R. Dey and F. M. Salemt, "Gate-variants of Gated Recurrent Unit (GRU) neural networks," *Midwest Symp. Circuits Syst.*, vol. 2017-Augus, no. 2, pp. 1597–1600, 2017, doi: 10.1109/MWSCAS.2017.8053243.

[95] R. Zhao, R. Yan, J. Wang, and K. Mao, "Learning to Monitor Machine Health with Convolutional Bi-Directional LSTM Networks," pp. 1–18, 2017, doi: 10.3390/s17020273.

[96] C. C. Aggarwal, "Transfer learning," *Data Classif. Algorithms Appl.*, pp. 657–665, 2014, doi: 10.1201/b17320.

[97] S. J. Pan and Q. Yang, "A Survey on Transfer Learning," *IEEE Trans. Knowl. Data Eng.*, vol. 194, pp. 781–789, 2009, doi: 10.1007/978-981-15-5971-6_83.

[98] T. Pan, J. Chen, J. Xie, Y. Chang, and Z. Zhou, "Intelligent fault

identification for industrial automation system via multi-scale convolutional generative adversarial network with partially labeled samples," *ISA Trans.*, vol. 101, pp. 379–389, 2020, doi: 10.1016/j.isatra.2020.01.014.

[99] L. Wen, L. Gao, and X. Li, "A New Deep Transfer Learning Based on Sparse Auto-Encoder for Fault Diagnosis," pp. 1–9, 2017.

[100] B. Yang, Y. Lei, F. Jia, and S. Xing, "An intelligent fault diagnosis approach based on transfer learning from laboratory bearings to locomotive bearings," *Mech. Syst. Signal Process.*, vol. 122, pp. 692–706, 2019, doi: 10.1016/j.ymssp.2018.12.051.

[101] A. Rzhetsky, J. G. Foster, I. T. Foster, and J. A. Evans, "Choosing experiments to accelerate collective discovery," *Proc. Natl. Acad. Sci. U. S. A.*, vol. 112, no. 47, pp. 14569–14574, 2015, doi: 10.1073/pnas.1509757112.

[102] L. Bolelli, Ş. Ertekin, and C. L. Giles, "Topic and trend detection in text collections using latent dirichlet allocation," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 5478 LNCS, pp. 776–780, 2009, doi: 10.1007/978-3-642-00958-7_84.

[103] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," 2008, doi: 10.1016/j.neucom.2018.05.040.

[104] H. Yin, Z. Li, J. Zuo, H. Liu, K. Yang, and F. Li, "Wasserstein Generative Adversarial Network and Convolutional Neural Network (WG-CNN) for Bearing Fault Diagnosis," vol. 2020, 2020.

[105] G. Colab *et al.*, "Late policy : late submission will not be marked (no matter in what reason)!!! Marking regulation . How to construct the training and

testing dataset (Very important) Where to download these datasets How to obtain top-1 accuracy for each given dataset," vol. 100, pp. 100–101.

[106] L. Biggio, M. Russi, S. Bigdeli, I. Kastanis, D. Giordano, and D. Gagar, "PHME Data Challenge," *Eur. Conf. Progn. Heal. Manag. Soc.*, 2021.

[107] G. P. Zhang and M. Qi, "Neural network forecasting for seasonal and trend time series," *Eur. J. Oper. Res.*, vol. 160, no. 2, pp. 501–514, 2005, doi: 10.1016/j.ejor.2003.08.037.

[108] A. Zien, N. Krämer, S. Sonnenburg, and G. Rätsch, "The feature importance ranking measure," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 5782 LNAI, no. PART 2, pp. 694–709, 2009, doi: 10.1007/978-3-642-04174-7_45.

[109] D. Bowen and L. Ungar, "Generalized SHAP: Generating multiple types of explanations in machine learning," 2020, [Online]. Available: http://arxiv.org/abs/2006.07155.

[110] Y. A. Alsariera, V. E. Adeyemo, A. O. Balogun, and A. K. Alazzawi, "AI Meta-Learners and Extra-Trees Algorithm for the Detection of Phishing Websites," *IEEE Access*, vol. 8, pp. 142532–142542, 2020, doi: 10.1109/ACCESS.2020.3013699.

[111] C. Chen, N. Wang, and M. Chen, "Prediction model of end-point phosphorus content in consteel electric furnace based on PCA-extra tree model," *ISIJ Int.*, vol. 61, no. 6, pp. 1908–1914, 2021, doi: 10.2355/isijinternational.ISIJINT-2020-615.

[112] N. Chakrabarty and S. Biswas, "Navo Minority Over-sampling Technique (NMOTe): A Consistent Performance Booster on Imbalanced Datasets," no.

June, 2020, doi: 10.36548/jei.2020.2.004.

[113] A. V. Dorogush, V. Ershov, and A. Gulin, “CatBoost : gradient boosting with categorical features support,” *arXiv*, pp. 1–7, 2018.

[114] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, “CatBoost : unbiased boosting with categorical features,” no. Section 4, pp. 1–11, 2018.

[115] G. Ke *et al.*, “LightGBM : A Highly Efficient Gradient Boosting Decision Tree,” in *31st Conference on Neural Information Processing Systems*, 2017, no. Nips, pp. 1–9.

[116] Y. J. Ong, N. Baracaldo, and H. Ludwig, “Adaptive histogram-based gradient boosted trees for federated learning,” *arXiv*, 2020.

[117] Y. Shi, J. Li, and Z. Li, “Gradient Boosting with Piece-Wise Linear Regression Trees,” *arXiv*, 2019.

[118] M. Abadi *et al.*, “TensorFlow : A System for Large-Scale Machine Learning This paper is included in the Proceedings of the TensorFlow : A system for large-scale machine learning,” 2016.

[119] H. Shafizadeh-Moghadam, A. Asghari, A. Tayyebi, and M. Taleai, “Coupling machine learning, tree-based and statistical models with cellular automata to simulate urban growth,” *Comput. Environ. Urban Syst.*, vol. 64, pp. 297–308, 2017, doi: 10.1016/j.compenvurbsys.2017.04.002.

[120] J. T. Townsend and W. Lafayette, “Theoretical analysis of an alphabetic confusion matrix,” vol. 9, 1971.

[121] S. Visa, B. Ramsay, A. Ralescu, and E. Van Der Knaap, “Confusion matrix-based feature selection,” *MAICS*, 2011.

[122] A. A. Salatino, “Early Detection of Research Trends,” *arXiv*, 2019, doi: 10.21954/ou.ro.00010698.

[123] D. V Carvalho, E. M. Pereira, and J. S. Cardoso, “Machine Learning Interpretability : A Survey on Methods and Metrics,” pp. 1–34, 2019, doi: 10.3390/electronics8080832.

[124] Y. Li and Y. Pan, “A novel ensemble deep learning model for stock prediction based on stock prices and news,” *Int. J. Data Sci. Anal.*, pp. 1–15, 2021, doi: 10.1007/s41060-021-00279-9.

[125] T. Rashed, J. R. Weeks, M. S. Gadalla, and A. G. Hill, “Revealing the anatomy of cities through spectral mixture analysis of multispectral satellite imagery: A case study of the Greater Cairo Region, Egypt,” *Geocarto Int.*, vol. 16, no. 4, pp. 7–18, 2001, doi: 10.1080/10106040108542210.

[126] P. A. Flach, P. E. F. Lach, and B. Ac, “The Geometry of ROC Space : Understanding Machine Learning Metrics through ROC Isometrics,” 2003.

[127] D. Chicco and G. Jurman, “The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation,” pp. 1–13, 2020.

Appendix

Data Pre-Processing Steps

```
Path1 = r'C:\Users\lenovo\Documents\PHM datasets\Totaldata'
df_normal = pd.DataFrame()
df_abnormal = pd.DataFrame()



for root, dirs, files in os.walk(Path1):
    for file in files:
        if file.startswith('class_ 0'):
            dft = make_array()
            dft['Class'] = 0
            df_normal = pd.concat([df_normal, dft], axis=0)
        elif file.startswith('class_ 2'):
            dft = make_array()
            dft['Class'] = 2
            df_abnormal = pd.concat([df_abnormal, dft], axis=0)
        elif file.startswith('class_ 3'):
            dft = make_array()
            dft['Class'] = 3
            df_abnormal = pd.concat([df_abnormal, dft], axis=0)
        elif file.startswith('class_ 5'):
            dft = make_array()
            dft['Class'] = 5
            df_abnormal = pd.concat([df_abnormal, dft], axis=0)
        elif file.startswith('class_ 7'):
            dft = make_array()
            dft['Class'] = 7
            df_abnormal = pd.concat([df_abnormal, dft], axis=0)
        elif file.startswith('class_ 9'):
            dft = make_array()
            dft['Class'] = 9
            df_abnormal = pd.concat([df_abnormal, dft], axis=0)
        elif file.startswith('class_ 4'):
```

```
            dft = make_array()
            dft['Class'] = 4
            df_abnormal = pd.concat([df_abnormal, dft], axis=0)
        elif file.startswith('class_11'):
            dft = make_array()
            dft['Class'] = 11
            df_abnormal = pd.concat([df_abnormal, dft], axis=0)
        elif file.startswith('class_12'):
            dft = make_array()
            dft['Class'] = 12
            df_abnormal = pd.concat([df_abnormal, dft], axis=0)


df_normal.to_pickle('DFnormal.pkl')
df_abnormal.to_pickle('DFabnormal.pkl')

df_normal = pd.read_pickle('DFnormal.pkl')
df_abnormal = pd.read_pickle('DFabnormal.pkl')


frames = [df_abnormal,df_normal]
dft = pd.concat(frames)

counts = dft.nunique()
to_del = [i for i,v in enumerate(counts) if v == 1]
print(to_del)

to_del_col = [('FuseOutsideOperationalSpace',3),
'LightBarrieActiveTaskDuration2', 'LightBarrierActiveTaskDuration1b',
'LightBarrierPassiveTaskDuration1b'
,'LightBarrierPassiveTaskDuration2','LightBarrierTaskDuration']

# drop useless columns
df_normal.drop(to_del_col, axis=1, inplace=True)
df_abnormal.drop(to_del_col, axis=1, inplace=True)
print(df_abnormal.shape)
print(df_normal.shape)
dft.drop(to_del_col, axis=1, inplace=True)

def nan_check(data):
    total = data.isnull().sum().sort_values(ascending= False)
    percent_1 = data.isnull().sum()/data.isnull().count()*100
```

```
    percent_2 = (np.round(percent_1,1)).sort_values(ascending = False)
    missing_data = pd.concat([total,percent_2],axis=1,keys=['Total','%'])
    return missing_data

#How much missing value:
dfcheckss = nan_check(dft)
high_sorted = dfcheckss.sort_values(["Total", "%"], ascending=False)
dist_df = high_sorted.reset_index(level=[0,1])
dist_df.plot(x ='Signal', y='%', kind = 'line',figsize=(30,30))
dist_df.head(34)


to_del = [('NumberFuseDetected',5)
,('FuseHeatSlopeOK',5)
,('NumberFuseDetected',6)
,('NumberFuseEstimated',5)
,('FeederBackgroundIlluminationIntensity',5)
,('IntensityTotalImage',5)
,('SharpnessImage',5)
,('NumberFuseDetected',2)
,('NumberFuseDetected',3)
,('NumberFuseDetected',4)
,('SharpnessImage',2)
,('SharpnessImage',3)
,('SharpnessImage',4)
,('SharpnessImage',6)
,('IntensityTotalImage',2)
,('IntensityTotalImage',3)
,('IntensityTotalImage',4)
,('IntensityTotalImage',6)
,('FeederBackgroundIlluminationIntensity',2)
,('FeederBackgroundIlluminationIntensity',3)
,('FeederBackgroundIlluminationIntensity',4)
,('FeederBackgroundIlluminationIntensity',6)
,('NumberFuseEstimated',2)
,('NumberFuseEstimated',3)
,('NumberFuseEstimated',4)
,('NumberFuseEstimated',6)
,('FuseHeatSlopeNOK',5)
,('FuseHeatSlopeOK',2)
,('FuseHeatSlopeOK',3)
,('FuseHeatSlopeOK',4)
```

```
,('FuseHeatSlopeOK',6)
,('FuseHeatSlope',5)
,('IntensityTotalThermoImage',5)
,('TemperatureThermoCam',5)]

df_normal.drop(to_del, axis=1, inplace=True)
df_abnormal.drop(to_del, axis=1, inplace=True)
print(df_abnormal.shape)
print(df_normal.shape)


df_abnormal = df_abnormal.fillna(0)
df_normal = df_normal.fillna(0)


frames = [df_abnormal,df_normal]
dftotal = pd.concat(frames)


# Multi index to single index:

mylevels= [dftotal.columns.levels[0],['0', '1', '2', '3', '4', '5', '6', '']]
dftotal.columns = dftotal.columns.set_levels(levels=mylevels)
dftotal.columns = ["_".join(pair) for pair in dftotal.columns]
dftotal.head()


dfclass = dftotal['Class_']
dftotal1 = dftotal.copy()
dftotal.drop('Class_', axis=1, inplace=True)
X = dftotal.values
y = dfclass

# summarize the shape of the dataset
print(X.shape, y.shape)


dfdiag1 = dftotal1.copy()
dfdiag = dftotal.copy()


#Scaler
from sklearn.preprocessing import QuantileTransformer
```

```
scaler = QuantileTransformer()
X = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,
random_state=1)
```

Model Implementations

Extra Tree Classifier

```
start_time = time.time()
model_ex = ExtraTreesClassifier().fit(X_train, y_train)

score_ex = model_ex.score(X_test,y_test)
Ypred_ex = model_ex.predict(X_test)
recall_ex = metrics.recall_score(y_test, Ypred_ex, average = 'macro')
precision_ex = metrics.precision_score(y_test, Ypred_ex, average = 'macro')
f1score_ex = metrics.f1_score(y_test, Ypred_ex, average='macro')
CohenKappa = metrics.cohen_kappa_score(y_test, Ypred_ex)

y_preb_probs = model_ex.predict_proba(X_test)
ROCAUC = roc_auc_score(y_test, y_preb_probs, average='macro', multi_class="ovr")
MATT = matthews_corrcoef(y_test, Ypred_ex)

print("accuracy : {}\n".format(score_ex),
      "precision : {}\n".format(precision_ex),
      "recall : {}\n".format(recall_ex),
      "f1 score : {}\n".format(f1score_ex),
      "kappa : {}\n".format(CohenKappa),
       'ROC : {}\n'.format(ROCAUC),
     'MATTEW  : {}\n'.format(MATT))


end_time = time.time()
print("Execution time: ", end_time - start_time,"secs")

confusion_matrix(y_test, Ypred_ex)
```

Bayesian Optimization Algorithm

```
def gbm_cl_bo(max_depth,
max_samples,min_samples_split,min_samples_leaf,max_leaf_nodes,n_jobs):
   params_gbm = {}
   params_gbm['max_depth'] = round(max_depth)
   #params_gbm['max_features'] = round(max_features)
   params_gbm['max_leaf_nodes'] = round(max_leaf_nodes)
   params_gbm['min_samples_leaf'] = round(min_samples_leaf)
   params_gbm['min_samples_split'] = round(min_samples_split)
   params_gbm['max_samples'] = round(max_samples)
   params_gbm['n_jobs'] = round(n_jobs)
   #params_gbm['n_estimators'] = round(n_estimators)
   scores = cross_val_score(ExtraTreesClassifier(**params_gbm),
                            X_train, y_train, scoring='accuracy', cv=3).mean()
   score = scores.mean()
   return score
# Run Bayesian Optimization
start = time.time()
params_gbm ={
   'max_depth' :(80,200),
   'n_jobs'  :(1,5),
   #'max_features':(2,10),
   'min_samples_split':(2,6),
   'max_leaf_nodes': (550,750),
   'min_samples_leaf':(2,6),
   #'n_estimators': (10,100),
   'max_samples':(70,160),

}
gbm_bo = BayesianOptimization(gbm_cl_bo, params_gbm, random_state=111)
gbm_bo.maximize(init_points=20, n_iter=4)
print('It takes %s minutes' % ((time.time() - start)/60))
```

XGBoost

```
start_time = time.time()
model_xb =
XGBClassifier(objective='multi:softmax',tree_method='approx').fit(X_train,
y_train)
```

```
score_xb = model_xb.score(X_test,y_test)
Ypred_xb = model_xb.predict(X_test)
recall_xb = metrics.recall_score(y_test, Ypred_xb, average = 'macro')
precision_xb = metrics.precision_score(y_test, Ypred_xb, average = 'macro')
f1score_xb = metrics.f1_score(y_test, Ypred_xb, average='macro')
CohenKappa = metrics.cohen_kappa_score(y_test, Ypred_xb)

y_preb_probs = model_xb.predict_proba(X_test)
ROCAUC = roc_auc_score(y_test, y_preb_probs, average='macro', multi_class="ovr")
MATT = matthews_corrcoef(y_test, Ypred_xb)

print("accuracy : {}\n".format(score_xb),
      "precision : {}\n".format(precision_xb),
      "recall : {}\n".format(recall_xb),
      "f1 score : {}\n".format(f1score_xb),
      "kappa : {}\n".format(CohenKappa),
       'ROC : {}\n'.format(ROCAUC),
     'MATTEW  : {}\n'.format(MATT))


end_time = time.time()
print("Execution time: ", end_time - start_time,"secs")

confusion_matrix(y_test, Ypred_xb)
```

CATBoost

```
start_time = time.time()

model_cb = CatBoostClassifier(border_count=None, verbose=False).fit(X_train,
y_train)

score_cb = model_cb.score(X_test,y_test)
Ypred_cb = model_cb.predict(X_test)
recall_cb = metrics.recall_score(y_test, Ypred_cb, average = 'macro')
precision_cb = metrics.precision_score(y_test, Ypred_cb, average = 'macro')
f1score_cb = metrics.f1_score(y_test, Ypred_cb, average='macro')
CohenKappa = metrics.cohen_kappa_score(y_test, Ypred_cb)

y_preb_probs = model_cb.predict_proba(X_test)
```

```
ROCAUC = roc_auc_score(y_test, y_preb_probs, average='macro', multi_class="ovr")
MATT = matthews_corrcoef(y_test, Ypred_cb)

print("accuracy : {}\n".format(score_cb),
      "precision : {}\n".format(precision_cb),
      "recall : {}\n".format(recall_cb),
      "f1 score : {}\n".format(f1score_cb),
      "kappa : {}\n".format(CohenKappa),
       'ROC : {}\n'.format(ROCAUC),
     'MATTEW  : {}\n'.format(MATT))


end_time = time.time()
print("Execution time: ", end_time - start_time,"secs")

confusion_matrix(y_test, Ypred_cb)
```

Hist Gradient Boosting

```
start_time = time.time()
model_hg =
HistGradientBoostingClassifier(loss='categorical_crossentropy').fit(X_train,
y_train)

score_hg = model_hg.score(X_test,y_test)
Ypred_hg = model_hg.predict(X_test)
recall_hg = metrics.recall_score(y_test, Ypred_hg, average = 'macro')
precision_hg = metrics.precision_score(y_test, Ypred_hg, average = 'macro')
f1score_hg = metrics.f1_score(y_test, Ypred_hg, average='macro')
CohenKappa = metrics.cohen_kappa_score(y_test, Ypred_hg)

y_preb_probs = model_hg.predict_proba(X_test)
ROCAUC = roc_auc_score(y_test, y_preb_probs, average='macro', multi_class="ovr")
MATT = matthews_corrcoef(y_test, Ypred_hg)

print("accuracy : {}\n".format(score_hg),
      "precision : {}\n".format(precision_hg),
      "recall : {}\n".format(recall_hg),
      "f1 score : {}\n".format(f1score_hg),
      "kappa : {}\n".format(CohenKappa),
       'ROC : {}\n'.format(ROCAUC),
```

```
    'MATTEW  : {}\n'.format(MATT))

end_time = time.time()
print("Execution time: ", end_time - start_time,"secs")

confusion_matrix(y_test, Ypred_hg)
```

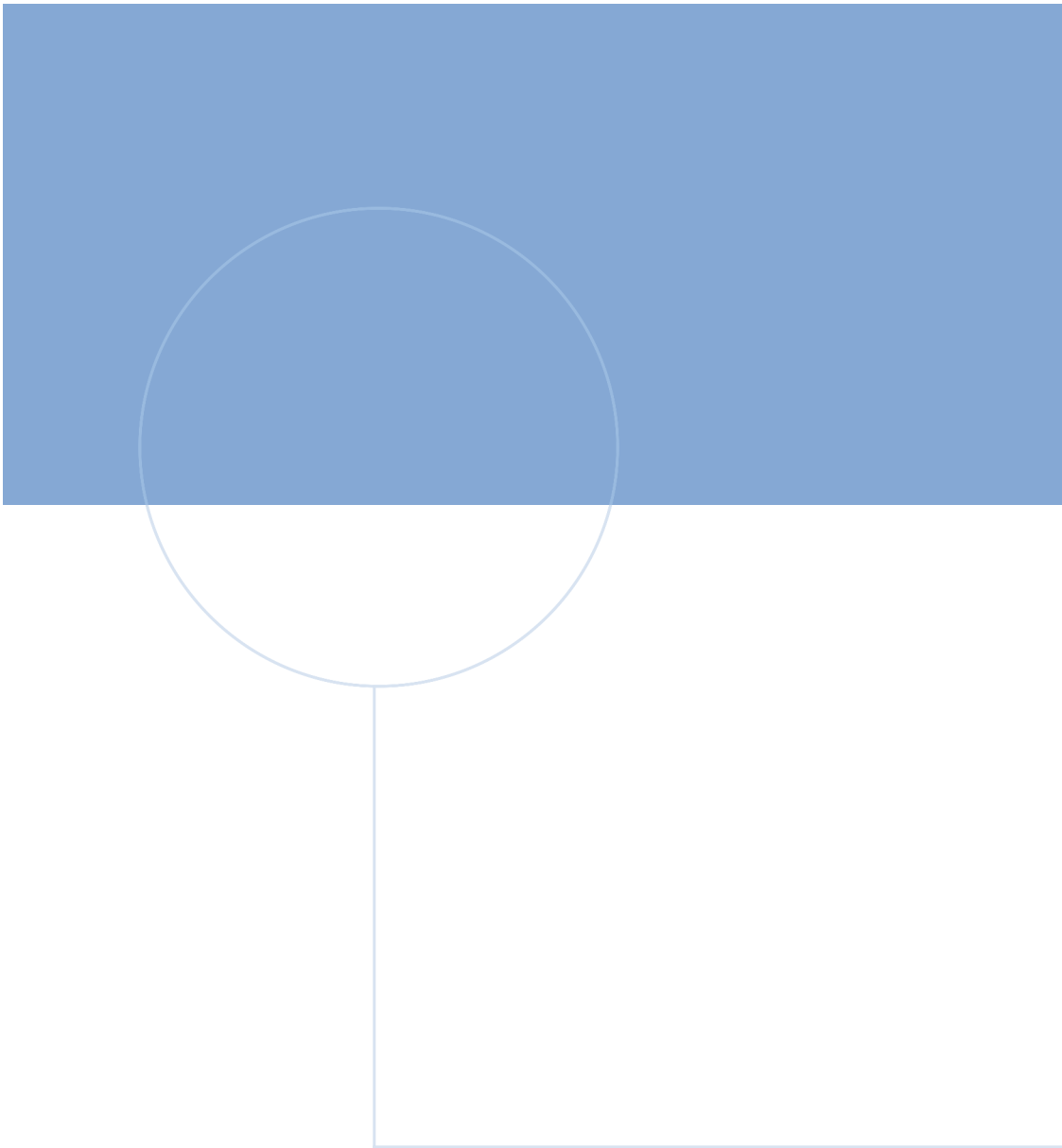
Deep Neural Network

```
X = np.load("/gdrive/MyDrive/Datasets/Projects/Mechanical Faults/My
Specialization Project/Final Data/X.npy")
Y = np.load("/gdrive/MyDrive/Datasets/Projects/Mechanical Faults/My
Specialization Project/Final Data/y.npy")

print(X.shape)
print(Y.shape)
print()
print(len(np.unique(Y)))
print(np.unique(Y))

from sklearn.preprocessing import LabelEncoder
X = X.astype('float32')
Y = LabelEncoder().fit_transform(Y.astype('str'))

print(len(np.unique(Y)))
print(np.unique(Y))

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X = scaler.fit_transform(X)

X_transform = X.reshape(X.shape[0], X.shape[1], 1)
print(X.shape)
print(X_transform.shape)

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,
random_state=1)

print(X_train.shape)
print(y_train.shape)
```

```
print(X_test.shape)
print(y_test.shape)

"""# Model Definition"""

# Commented out IPython magic to ensure Python compatibility.
# %cd /gdrive/MyDrive/Datasets/Projects/Mechanical Faults/Models

"""## Model Implementation"""

keras.backend.clear_session()

model = Sequential()
model.add(Dense(50, activation='relu'))
model.add(Dense(100, activation='relu'))
model.add(Dense(200, activation='relu'))
model.add(Dense(300, activation='relu'))
model.add(Dense(200, activation='relu'))
model.add(Dense(100, activation='relu'))
model.add(Dense(50, activation='relu'))
model.add(Dense(9, activation='softmax'))

opt = RMSprop(learning_rate=0.0001, momentum=0.1, centered=False)

model.compile(loss='sparse_categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])

es = EarlyStopping(monitor='val_loss', verbose=1, patience=20)
mc = ModelCheckpoint('final_model.h5', monitor='val_loss', verbose=1,
save_best_only=True)

#model.summary()

history = model.fit(X_train, y_train,
                    validation_split=0.2,
                    batch_size=None,
                    callbacks=[es, mc],
                    epochs=150)

plt.style.use('ggplot')
plt.plot(history.history['accuracy'])
```

```
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
#plt.ylim([0, 1])
plt.legend(['Train', 'Validation'], loc='lower right')
plt.show()

plt.style.use('ggplot')
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss (Sparse Categorical Crossentropy)')
plt.ylabel('Sparse Categorical Crossentropy')
plt.xlabel('Epoch')
#plt.ylim([0, 1.5])
plt.legend(['Train', 'Validation'], loc='upper right')
plt.show()

"""## Model Test"""

model = load_model('final_model.h5')

pred = model.predict(X_test)
eval = model.evaluate(X_test, y_test)

recall_dnn = metrics.recall_score(y_test, np.argmax(pred, axis=-1), average =
'macro')
precision_dnn = metrics.precision_score(y_test, np.argmax(pred, axis=-1), average
= 'macro')
f1score_dnn = metrics.f1_score(y_test, np.argmax(pred, axis=-1), average='macro')
CohenKappa = metrics.cohen_kappa_score(y_test, np.argmax(pred, axis=-1))
ROCAUC = roc_auc_score(y_test, np.argmax(pred, axis=-1, average='macro',
multi_class="ovr"))
MATT = matthews_corrcoef(y_test, np.argmax(pred, axis=-1))

print()
print("accuracy  :  {}".format(round(eval[1], 7)))
print("precision :  {}".format(round(precision_dnn, 7)))
print("recall    :  {}".format(round(recall_dnn, 7)))
print("f1 score  :  {}".format(round(f1score_dnn, 7)))
print("Kappa     :  {}".format(round(CohenKappa, 7)))
print("ROCAUC    :  {}".format(round(ROCAUC, 7)))
```

```
print("MATT      :  {}".format(round(MATT, 7)))
print()
confusion_matrix(y_test, np.argmax(pred, axis=-1))

model.summary()
```



NTNU

Kunnskap for en bedre verden