Order N°……/Faculty/UMBB/2023

People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
University M'Hamed BOUGARA – Boumerdes

Faculty of Hydrocarbons and Chemistry

Final Year Thesis Presented in Partial Fulfillment of the Requirements for the Degree of:

MASTER

In **Automation of Industrial Processes**

Option: **Automatic control**

Presented by:

BENSMAINE Abdelkrim **HAMMA Hichem**

Title

LSTM-Autoencoder deep learning model for predictive maintenance of an electric motor

Jury Members:

Mr. BOUMEDIENE Mohamed Said	MCA	FHC	President
Mr. HAMADACHE Mohamed	MCB	FHC	Examiner
Mr. YOUSSEF Tewfik	MCB	FHC	Examiner
Ms. LACHEKHAB Fadhila	MCB	FHC	Supervisor

academic year: 2022/2023

People’s Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
University M’Hamed BOUGARA – Boumerdes

Faculty of Hydrocarbons and Chemistry

Department: Automation of Industrial Processes
Option: Automatic Control

Final Year Thesis Presented in Partial Fulfillment of the Requirements for the Degree of:

MASTER

Title

LSTM-Autoencoder deep learning model for predictive maintenance of an electric motor

Presented by:

BENSMAINE Abdelkrim
HAMMA Hichem

Favorable assessment of the Supervisor:

F. LACHEKHAB signature

Favorable opinion of the President of the Jury:

Name **signature**

stamp and signature

Acknowledgements

First and foremost, we are grateful and thankful to Allah, the most gracious, and the most merciful for helping us complete this modest work.

We would also like to extend our thanks to our supervisor, Ms. F. Lachekhab, for her guidance, support, and valuable insights throughout the entire project. Her expertise and commitment were instrumental in shaping the direction of our research and pushing us to achieve the best possible outcomes.

Furthermore, we are appreciative to our friends and family for their unwavering support and encouragement throughout this journey. Their belief in us has been a constant source of motivation.

To all those mentioned above and anyone else who has directly or indirectly contributed to the completion of this project, we extend our heartfelt thanks. Your support and involvement have played a crucial role in our personal and academic growth.

Sincerely,

BENSMAINE Abdelkrim

HAMMA Hichem

Dedication

To my mother & father whose love & support for me was unlimited and their care and warmth never left me alone, I thank you from the depths of my heart for making me the person I am today. May Allah protect you.

To my siblings & family, you always have been there for me. Thank you greatly.

To my friends who were a second family through this journey. Thank you for everything.

-Abdelkrim-

To my mother and father, whose unwavering support, encouragement, and sacrifices have been the foundation of my journey. Your presence, your guidance, and your unconditional love have shaped the person I am today. I am forever grateful for the countless sacrifices you have made to provide me with opportunities and for instilling in me the values of hard work and perseverance. This achievement would not have been possible without you. Thank you for everything.

To my siblings and family, your simple presence always was heartwarming, thank you.

To friends who became family through this valuable journey, thank you.

-Hichem-

Contents

List of Figures

List of Acronyms

RM	Reactive Maintenance
PM	Preventive Maintenance
PdM	Predictive Maintenance
AI	Artificial Intelligence
ML	Machine Learning
RL	Reinforcement Learning
Q-Learning	Quality Learning
ACL	Actor Critic Learning
DL	Deep Learning
ANN	Artificial Neural Network
DNN	Deep Neural Network
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
FRNN	Fully connected Recurrent Neural Network
GRU	Gated Recurrent Unit
LSTM	Long Short-Term Memory
AE	Autoencoder
GPU	Graphic Processing Unit
TPU	Tensor Processing Unit
ReLU	Rectified Linear Unit
NLP	Natural Language Processing
MFS	Machinery Fault Simulator
ABVT	Alignment-Balance-Vibration Test
MSE	Mean Squared Error
CSV	Comma-Separated Value

General introduction

Since the first industrial revolution in the late 18th century, industrial maintenance techniques have evolved over time to address the challenges of equipment reliability and performance in industrial settings. Initially, the predominant approach was reactive maintenance, also known as "breakdown maintenance". This involved waiting for equipment failures to occur and then taking corrective actions to fix the issues. While this approach was simple and cost-effective in the short term, it often resulted in significant production losses, safety hazards, and higher repair costs.

As industries became more complex and downtime costs increased, preventive maintenance emerged as a more proactive strategy, involving routine inspections and maintenance tasks based on predetermined schedules. This approach aimed to prevent unexpected failures by addressing known wear and tear issues. While preventive maintenance reduced unplanned downtime to some extent, it was not always efficient and often led to unnecessary maintenance activities and associated costs.

In recent years, with advancements in technology and the rising implementation of Artificial Intelligence techniques in the industrial sector, predictive maintenance (PdM) has gained prominence. This approach utilizes real-time data from sensors, monitoring systems, and predictive algorithms to assess equipment condition, identify potential failures, and schedule maintenance activities accordingly. By adopting this approach, organizations can optimize maintenance schedules, reduce costs, maximize equipment uptime, and enhance overall operational efficiency.

Furthermore, emerging technologies such as Artificial Intelligence, Artificial Neural Networks (ANNs), and Deep Learning (DL) methods are being integrated into industrial maintenance practices. These technologies enable more advanced data processing, anomaly detection, and predictive modeling, leading to more accurate predictions and optimized maintenance strategies.

PdM is a superior and complex maintenance technique, it requires large amounts of data to establish a pattern and predict the failures before occurring, which is why deep learning techniques are an essential part of PdM. One of the most common DL techniques is the Long Short-Term Memory (LSTM) architecture, which is a type of recurrent neural network (RNN) that is capable of modeling and predicting sequential data. In PdM, LSTM can be used to analyze time-series data from sensors or monitoring systems to detect patterns, trends, and anomalies. By learning the temporal dependencies in the data, LSTM models can make predictions about future equipment behavior and identify potential faults or failures.

In this thesis, we divided our work into four Chapters:

- In the first Chapter, we will talk about industrial maintenance approaches, especially PdM, and how it can improve the performance of industrial systems and machinery. Furthermore, we will introduce AI and its different subfields, including Machine Learning, ANNs and DL techniques.
- In the second Chapter, we will dive in the LSTM and Autoencoder architectures used in our model, explaining how these architectures function and how they are utilized in PdM and anomaly detection.
- In the third Chapter, we will introduce our Autoencoder model, the program used to create it, the data-base used to train it, and visualize its performance.
- Finally, in the fourth and last Chapter, we will present our LSTM-Autoencoder model, which is a combination of LSTM and Autoencoders that leverage the features of both architectures. We will explain this final model and compare its performance with the previous regular Autoencoder model introduced in Chapter three, and eventually make our conclusions.

Chapter I

Overview on Predictive Maintenance and AI

Chapter I
Overview on Predictive Maintenance and AI

I.1 Introduction

Predictive maintenance (PdM) approaches have recently been widely used in industries for managing the health status of industrial equipment as smart systems, machine learning (ML), and deep learning (DL), within artificial intelligence (AI) have emerged. As a result of the development and increasing popularity of these DL algorithms, it is now possible to gather enormous amounts of operational and process conditions data generated from various pieces of equipment and use the data to make an automated fault detection, diagnosis, and most importantly, a prognosis to reduce and predict downtime and increase the utilization rate of the components [1].

In this chapter, we are going to talk about Predictive Maintenance, Artificial Intelligence, and all its subfields in a general manner, including the deep learning method used in our model, and how it is related to the PdM approach.

I.2 Industrial Maintenance

Industrial maintenance, refers to the activities and processes carried out to ensure the optimal functioning, reliability, and longevity of industrial machinery, equipment, and systems within a manufacturing or production facility. It involves a range of tasks aimed at preventing equipment failures, minimizing downtime, and maximizing productivity.

Over the years, the field of industrial maintenance has undergone remarkable development, fueled by technological advancements, it shifted towards more proactive and efficient practices. Initially, maintenance practices were reactive, with equipment failures addressed as they occurred. As industries grew, preventive maintenance emerged, involving scheduled inspections and component replacements based on predetermined time intervals. However, this approach lacked precision.

The advent of sensor technology and data analysis led to the rise of condition-based maintenance, which allowed monitoring equipment health in real-time and performing maintenance based on actual conditions. Building on this, predictive maintenance emerged, utilizing advanced analytics and machine learning to predict failures and enable proactive planning.

In summary, the evolution of industrial maintenance has transitioned from reactive to more proactive, data-driven, and technology-enabled approaches. Integrating these approaches enhances equipment reliability, reduces downtime, and achieves higher operational efficiency [42, 43, 44].

Generally, maintenance techniques can be resumed in three categories:

- Reactive maintenance (RM)
- Preventive maintenance (PM)
- Predictive maintenance (PdM)

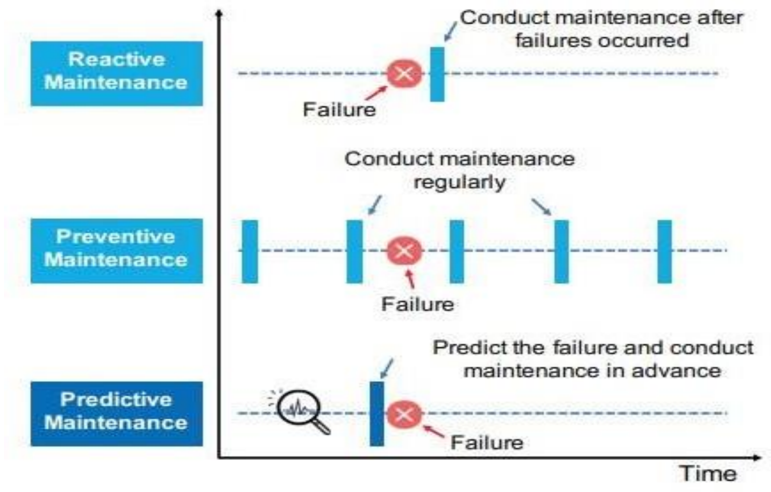

Reactive Maintenance
Conduct maintenance after failures occurred
Failure
Conduct maintenance regularly
Preventive Maintenance
Failure
Predict the failure and conduct maintenance in advance
Predictive Maintenance
Failure
Time

Figure I.1 Maintenance plans of RM, PM, and PdM [44]

I.2.1 Reactive Maintenance

In the early stages of industrialization, maintenance was predominantly reactive. Equipment failures were addressed as they occurred, resulting in unplanned downtime, productivity losses, and higher maintenance costs.

Also known as Breakdown Maintenance, Reactive Maintenance is a maintenance technique where repairs or replacements are carried out in response to equipment failures or breakdowns. In this reactive paradigm, maintenance actions are taken only after a failure occurs, rather than being planned or anticipated in advance. When a piece of equipment stops functioning or exhibits a malfunction, maintenance personnel are mobilized to address the issue and restore it to working condition [42].

RM is characterized by its unplanned and reactive nature, without any preventive measures in place, equipment failures can occur suddenly and disrupt operations, resulting in delays, reduced output, and potential safety risks. The focus of reactive maintenance is to resolve the immediate problem and get the equipment back up and running as quickly as possible.

While reactive maintenance may be necessary in certain situations, it is generally regarded as a less desirable approach compared to proactive maintenance strategies. It can be costlier due to the urgent nature of repairs, the need for rush orders of replacement parts, and the potential for collateral damage caused by the failure. Additionally, unplanned downtime can result in missed production targets, customer dissatisfaction, and increased overtime costs.

Reactive maintenance can be suitable for non-critical equipment with minimal impact on overall operations or when the cost of implementing preventive or predictive maintenance strategies outweighs the potential losses from reactive maintenance.

However, in modern industrial settings, organizations strive to minimize reactive maintenance by adopting more proactive and preventive maintenance approaches. These proactive strategies, such as preventive maintenance or predictive maintenance, aim to identify and address potential issues before they lead to equipment failures, resulting in improved reliability, increased equipment lifespan, and reduced operational disruptions.

I.2.2 Preventive Maintenance

As industries grew, and with the several downsides of RM, the concept of preventive maintenance emerged.

Preventive maintenance is a systematic approach to maintenance that focuses on scheduled inspections to prevent equipment failures and maximize operational efficiency. In preventive maintenance, maintenance activities are planned and carried out before any signs of failure or breakdown occur. The goal is to maintain equipment in optimal condition, extend its lifespan, and minimize the risk of unexpected failures [44].

PM strategies typically involve routine inspections, lubrication, cleaning, and adjustments based on manufacturer recommendations, industry best practices, and historical data. These scheduled maintenance tasks are performed at predetermined intervals, such as daily, weekly, monthly, or annually, depending on the equipment and its operating conditions.

By adhering to these schedules, potential issues can be identified early, allowing for timely intervention and minimizing the likelihood of major breakdowns or malfunctions.

The benefits of preventive maintenance are numerous. By conducting regular inspections and addressing minor issues promptly, the likelihood of equipment failure and unscheduled downtime is significantly reduced. This leads to improved productivity, increased operational efficiency, and reduced costs associated with emergency repairs and production interruptions. Additionally, preventive maintenance helps identify and replace worn-out or faulty components before they cause further damage or affect the performance of other interconnected systems.

To effectively implement a preventive maintenance program, organizations often maintain detailed maintenance records, including equipment histories, maintenance schedules, and task checklists. These records help track maintenance activities, identify trends, and plan future maintenance tasks.

Preventive maintenance is particularly suitable for critical equipment, machinery, and systems that have a significant impact on production, safety, or compliance. By investing in regular maintenance and inspections, organizations can ensure equipment reliability, enhance workplace safety, comply with regulatory requirements, and achieve higher levels of customer satisfaction.

While preventive maintenance offers numerous benefits, it also has several downsides that must be taken into consideration [43].

Firstly, implementing a preventive maintenance program incurs costs, as resources must be allocated for regular inspections, servicing, and component replacements. Over-maintenance is another concern, as excessive or unnecessary maintenance tasks can lead to increased costs and inefficiencies. Additionally, preventive maintenance activities can disrupt operations, requiring equipment to be temporarily taken offline and impacting production schedules. Lastly, despite regular maintenance, unforeseen failures can still occur, and relying solely on preventive measures may not address these unexpected events.

For these reasons, another maintenance technique had to be developed to improve the effectiveness and reliability of the machinery. Thus, Predictive maintenance emerged.

I.2.3 Predictive Maintenance

Predictive Maintenance (PdM) is a maintenance strategy that uses data analysis tools and techniques to predict when maintenance or repair work on a specific piece of equipment or machinery is required. The idea behind predictive maintenance is to avoid unscheduled downtime and cut maintenance costs by performing maintenance work only when it is necessary [15].

Predictive maintenance relies on a variety of methods, including data analytics, machine learning, and artificial intelligence, to identify patterns and trends in equipment performance data. PdM algorithms can detect early warning signs of equipment failure or deterioration by analyzing data from sensors, maintenance logs, and other sources, and then provide recommendations for maintenance actions to prevent or at least mitigate the issues detected [16].

The foundation of predictive maintenance lies in collecting and analyzing relevant data such as vibration, temperature, pressure, and performance metrics. Advanced analytics algorithms process the data to identify trends, deviations from normal operating conditions, and indicators of potential failure. Machine learning techniques can be applied to these datasets to develop models that can predict failure probabilities and estimate remaining useful life.

By analyzing historical data and correlating it with real-time data (from sensors for example), predictive maintenance algorithms can detect early signs of equipment degradation, wear, or impending failure. This enables maintenance teams to intervene proactively, performing targeted maintenance tasks such as component replacements, or adjustments when the data indicates an increased risk of failure. This approach minimizes the chances of unexpected breakdowns, reduces downtime, and optimizes the utilization of maintenance resources.

The benefits of predictive maintenance are numerous. By addressing maintenance needs before failures occur, organizations can significantly reduce unplanned downtime, which can be costly and disruptive. PdM enables the scheduling of maintenance activities during planned downtime or low-demand periods, minimizing the impact on production schedules. The ability to anticipate failures and plan maintenance tasks in advance also reduces the need for emergency repairs and rush orders for replacement parts, resulting in cost savings.

Furthermore, predictive maintenance allows for better asset management by optimizing the lifespan of critical equipment. By identifying and addressing potential issues early, organizations can extend the life of their assets, postpone costly capital investments, and maximize return on investment. Predictive maintenance also supports condition-based optimization, where maintenance intervals are adjusted based on actual equipment health, ensuring that maintenance efforts are performed when needed, rather than on a fixed schedule.

However, implementing a predictive maintenance program requires a robust data infrastructure, reliable sensors, and sophisticated analytics capabilities. It also requires skilled personnel who can interpret the data and act upon the insights provided by predictive models. Additionally, the success of predictive maintenance relies on the availability of historical data and the continuous monitoring of equipment health. Therefore, organizations need to invest in data collection systems, data storage, and analytics tools to support predictive maintenance initiatives effectively.

In summary, predictive maintenance enables organizations to move from reactive or time-based maintenance approaches to a proactive and data-driven strategy. By leveraging real-time data analytics and machine learning, predictive maintenance allows for early detection of equipment failures, targeted maintenance interventions, and optimized resource utilization.

I.3 Artificial Intelligence

Artificial Intelligence (AI) is a rapidly evolving field that has the potential to revolutionize the way we live and work. In recent years, it has been applied to a wide range of industries and has shown promising results in improving efficiency, accuracy, and decision-making. According to a recent report by McKinsey, AI could contribute up to $13 trillion to the global economy by 2030 [2].

At its core, AI is a discipline of study that focuses on creating intelligent machines that can perform tasks that typically require human intelligence, such as learning, problem-solving, decision-making, and perception. The goal of AI is to develop systems that can operate autonomously, adapt to new situations, and interact with humans and the environment in a natural and seamless way [3].

From an industrial perspective, AI can be defined as the brain that allows a system to detect its environment, interpret the data it collects, solve complicated issues, and learn from experience [4].

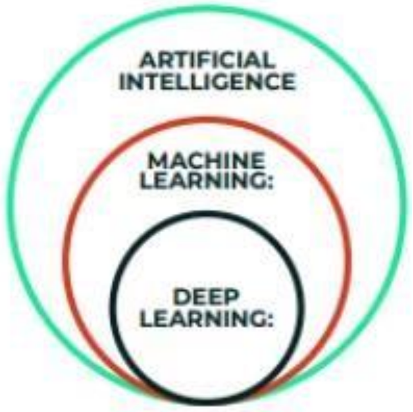


Figure I.2 Representation of Artificial Intelligence and its subfields [5]

I.3.1 Machine Learning

One of the key drivers of Artificial Intelligence is machine learning (ML), a subset of AI that focuses on developing algorithms that can learn from data and improve their performance over time.

Machine learning algorithms organize the data, learn from it, gather insights, and make predictions based on the information it analyzed without the need for additional explicit programming. Training a model with data and after that using the model to predict any new data is the concern of Machine Learning [5].

One of the significant advantages of machine learning is its ability to handle complex and large-scale datasets. By processing vast amounts of data, machine learning algorithms can uncover intricate patterns and relationships that may not be apparent to humans. This enables applications in various domains, such as image and speech recognition, natural language processing, recommendation systems, fraud detection, and autonomous vehicles.

ML is a powerful field of study that enables computers to learn from data, discover patterns, and make predictions or decisions. Through the use of sophisticated algorithms and statistical techniques, machine learning has the potential to transform industries and solve complex problems. As technology continues to advance, machine learning will play a crucial role in shaping the future of artificial intelligence and driving innovation in various domains.

Machine learning techniques can be subdivided into supervised, unsupervised, semi-supervised, and reinforcement learning [6, 7].

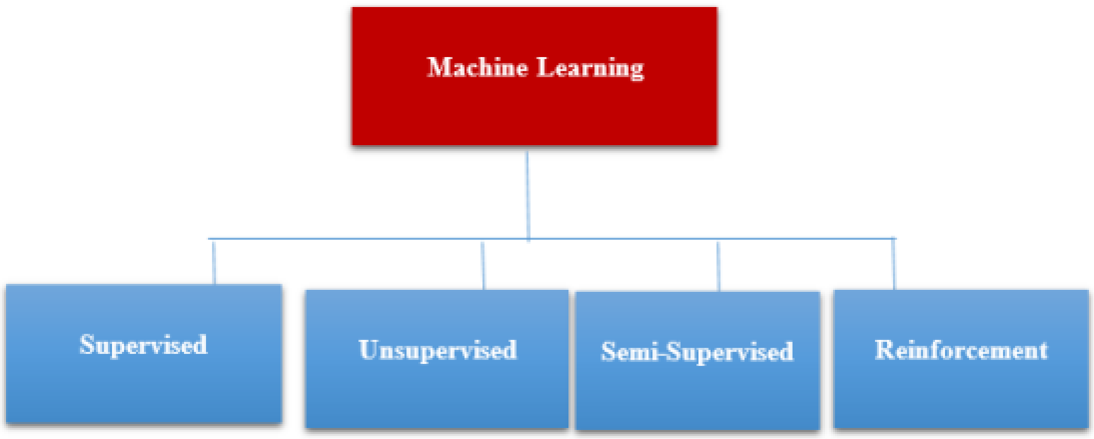


Figure I.3 Components of Machine Learning [5]

I.3.1.1 Supervised Learning

Supervised learning is a machine learning technique where the algorithm learns from labeled data, with each data point having corresponding input features and known output labels. The goal of supervised learning is to build a predictive model that can accurately map input data to the correct output labels based on the provided training examples [47].

The process of supervised learning begins with the collection of a labeled dataset, where each data instance is associated with a known output value. This dataset is then divided into two parts: the training set and the test set. The training set is used to train the model by presenting it with input features and their corresponding labels. The model learns from the training data by adjusting its internal parameters or weights based on the observed input-output relationships. The objective is to minimize the difference between the predicted output and the actual label for each training example.

Once the model is trained, it is evaluated using the test set, which consists of unseen data with known labels. The model's performance is assessed by comparing its predicted outputs with the true labels.

The main advantage of supervised learning is its ability to make accurate predictions or classifications based on labeled data. It is widely used in various applications, including spam detection, sentiment analysis, image recognition, speech recognition, and medical diagnosis. Supervised learning models can also be extended to handle multiclass classification problems and support probabilistic predictions, providing valuable insights for decision-making.

However, supervised learning also has limitations. It heavily relies on the availability of labeled data, which can be expensive and time-consuming to obtain. An insufficient or biased dataset may lead to inaccurate models and poor generalization of unseen data. Additionally, supervised learning models may struggle when faced with data that falls outside the range of the training examples, making them sensitive to outliers and noise [47].

In summary, supervised learning is a powerful machine learning approach that leverages labeled data to build predictive models. It enables accurate predictions or classifications by learning from observed input-output relationships. While it has its limitations, supervised learning has proven to be valuable in solving a wide range of real-world problems and continues to be a fundamental technique in the field of machine learning.

I.3.1.2 Unsupervised Learning

In unsupervised learning, the data is not labelled, which means that the ML model aims to discover unknown patterns in the data, by searching for similarities between the data points for example. Algorithms are therefore formulated such that they can find patterns and structures in the data on their own [48].

The process of unsupervised learning begins with collecting a dataset consisting of input features without corresponding output labels. The goal is to find meaningful representations or groupings within the data. Clustering is one common technique in unsupervised learning, where similar data points are grouped based on their inherent similarities. Clustering algorithms, such as k-means, hierarchical clustering, and DBSCAN, are used to identify clusters and partition the data accordingly.

Another key approach in unsupervised learning is dimensionality reduction, which aims to reduce the number of input features while preserving important information. This helps in visualizing high-dimensional data and extracting relevant features. Autoencoders are commonly used methods for dimensionality reduction.

Unsupervised learning algorithms can also be used for anomaly detection, where the goal is to identify unusual or abnormal data points that deviate significantly from the norm. By learning the regular patterns in the data, unsupervised algorithms can detect outliers or anomalies that may indicate potential fraud, errors, or unusual behavior.

Evaluation in unsupervised learning is more challenging than in supervised learning since there are no predefined output labels to compare against. Instead, the quality of unsupervised learning algorithms is assessed based on the coherence and meaningfulness of the discovered patterns, the compactness of clusters, or the ability to separate anomalies from normal data.

However, unsupervised learning has its challenges. Since there are no ground truth labels, evaluating the performance of unsupervised algorithms can be subjective and depend on domain knowledge. The algorithms heavily rely on the quality and representativeness of the data, making it crucial to preprocess and clean the data appropriately. Additionally, unsupervised learning algorithms can be computationally expensive, especially when dealing with large-scale datasets or complex structures.

In summary, unsupervised learning is a valuable approach in machine learning that allows for exploring and extracting patterns from unlabeled data. By uncovering hidden structures and relationships, unsupervised learning algorithms provide insights, aid in data exploration, and serve as a foundation for various downstream tasks.

I.3.1.3 Semi-Supervised Learning

Semi-supervised learning is a branch of machine learning that combines elements of both supervised and unsupervised learning. It deals with datasets that contain a small portion of labeled data and a larger portion of unlabeled data. The goal of semi-supervised learning is to leverage the limited labeled data together with the unlabeled data to improve the model's performance and generalization [49].

The process of semi-supervised learning begins by partitioning the available data into labeled and unlabeled subsets. The labeled data consists of input features along with their corresponding output labels. The unlabeled data, on the other hand, contains input features without any associated labels. The labeled data is used to train a model using supervised learning techniques, while the unlabeled data is leveraged to enhance the model's performance.

Semi-supervised learning algorithms often incorporate unsupervised learning methods to exploit the unlabeled data. By leveraging the inherent structure and patterns within the unlabeled data, the algorithms aim to improve the model's ability to generalize to unseen data. Unsupervised learning techniques such as clustering, dimensionality reduction, or generative models can be used to extract additional information from the unlabeled data.

One common approach in semi-supervised learning is to use the unlabeled data to create a smoother decision boundary or to estimate the underlying data distribution. By considering the relationships and similarities among the unlabeled data points, the model can make more informed predictions for new, unseen instances.

Semi-supervised learning is particularly useful in scenarios where obtaining labeled data is costly, time-consuming, or difficult. By making effective use of a small labeled dataset in conjunction with a larger unlabeled dataset, semi-supervised learning can achieve comparable or even superior performance to supervised learning approaches that rely solely on labeled data.

In conclusion, semi-supervised learning is a powerful approach that combines elements of supervised and unsupervised learning to leverage both labeled and unlabeled data. By effectively utilizing the unlabeled data, semi-supervised learning algorithms can improve the model's performance and generalization, particularly in scenarios where obtaining labeled data is limited or expensive. Despite its challenges, semi-supervised learning continues to be an active area of research, driving advancements in machine learning and expanding the range of problems that can be addressed.

I.3.1.4 Reinforcement Learning

In a reinforcement learning (RL) system, instead of providing input and output pairs, we describe the current state of the system, specify a goal, provide a list of allowable actions and their environmental constraints for their outcomes, and let the ML model experience the process of achieving the goal by itself using the principle of trial and error to maximize a reward [8].

An agent interacts with an environment sequentially. At each step, the agent observes the current state of the environment and takes action. The environment responds by transitioning to a new state and providing feedback in the form of a reward signal, which indicates the desirability of the agent's action. The goal of the agent is to learn a policy (a mapping from states to actions) that maximizes the expected cumulative reward over time [41].

One key aspect of reinforcement learning is the trade-off between exploration and exploitation. Initially, the agent explores different actions and learns about the environment. As it gathers more knowledge, it shifts towards exploiting its current knowledge to maximize rewards.

Reinforcement learning has been successfully applied to various domains, such as robotics, game playing, autonomous vehicles, recommendation systems, and resource management.

RL encompasses a wide range of algorithms that can be used depending on the problem at hand, the most common ones are Q-Learning and Actor-Critic Learning (ACL).

- **Q-Learning** is a model-free algorithm used in reinforcement learning to learn the optimal action-value function, often referred to as the Q-function. The Q-function represents the expected cumulative reward for taking a particular action in a given state and following a specific policy. The Q-Learning algorithm iteratively updates the Q-values based on the observed rewards and the estimated future rewards. It uses a technique called Temporal Difference learning, which calculates the difference between the estimated Q-value and the observed reward to update the Q-value. By repeatedly interacting with the environment and updating the Q-values, the agent can learn the optimal policy that maximizes the cumulative reward [45].

- **Actor-Critic Learning (ACL)** is an approach of RL that combines elements of both policy-based and value-based methods. It utilizes two components: an actor and a critic. The actor is responsible for learning and selecting actions based on the current policy. It explores the environment, take actions, and gathers experiences. The critic, on the other hand, evaluates the actions taken by the actor and provides feedback in the form of a value function or Q-values. The actor-critic architecture allows for continuous learning and policy improvement. The actor uses the feedback from the critic to update its policy, while the critic uses the observed rewards to update its value estimates. This way, the actor-critic algorithm can learn both the best actions to take and the value of those actions [46].

In summary, Q-Learning is a value-based algorithm that learns the optimal action-value function, while Actor-Critic Learning combines policy-based and value-based methods to learn both the policy and the value function simultaneously. Both approaches have been widely used in reinforcement learning and have contributed to many successful applications.

I.3.2 Artificial Neural Network

Artificial Neural Network (ANN) is a type of ML inspired by the principle of information processing in biological systems, ANNs consist of mathematical representations of connected processing units called artificial neurons [8].

Like synapses in a brain, each connection between neurons transmits signals whose strength can be amplified or attenuated by a weight that is continuously adjusted during the learning process. Signals are only processed by subsequent neurons if a certain threshold is exceeded as determined by an activation function.

Typically, neurons are organized into networks with different layers. An input layer usually receives the data input and an output layer produces the ultimate result. In between, there are zero or more hidden layers that are responsible for learning a non-linear mapping between input and output.

“Feed-forward” is the first, most common and simplest architecture. It is formed by stacked neurons creating layers, where all the neurons of a layer are connected to all the neurons of the next layer by feeding their output to others’ input. However, there are no connections to neurons of previous layers or among neurons of the same layer [11].

Artificial neural networks are of particular interest since their flexible structure allows them to be modified for a wide variety of contexts across all types of ML, therefore, ANNs can be referred to as “Shallow” or “Deep” depending on the number of hidden layers it contains.

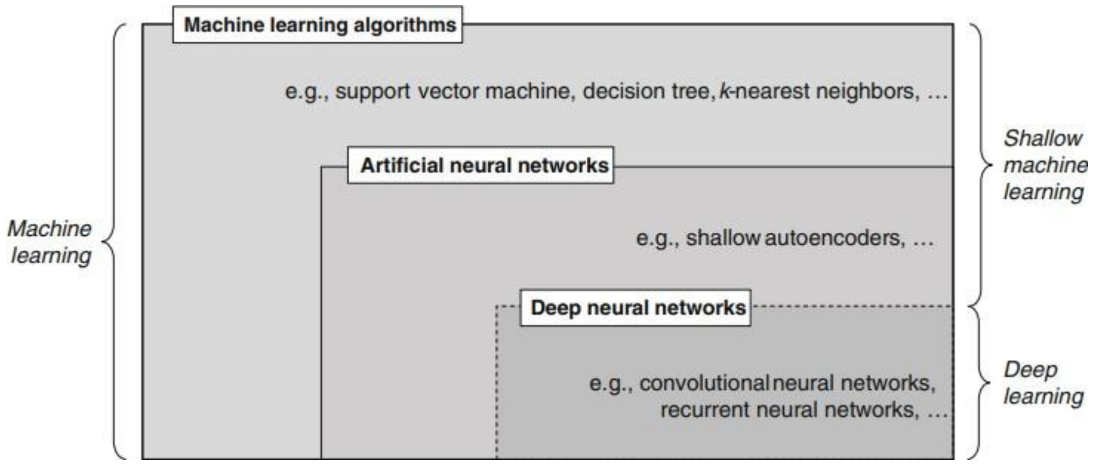

Machine learning algorithms
e.g., support vector machine, decision tree, k-nearest neighbors, ...
Artificial neural networks
e.g., shallow autoencoders, ...
Deep neural networks
e.g., convolutional neural networks, recurrent neural networks, ...
Machine learning
Shallow machine learning
Deep learning

Figure I.4 Diagram of ML classes [8]

I.3.3 Deep Learning

Deep learning is a subset of machine learning that focuses on the development and application of artificial neural networks with multiple layers, known as deep neural networks. It aims to enable computers to learn and make predictions or decisions by mimicking the structure and function of the human brain. Deep learning has gained significant attention and popularity due to its remarkable ability to automatically learn hierarchical representations from raw data, leading to a state-of-the-art performance in various domains [50].

At the core of deep learning are artificial neural networks, which consist of interconnected nodes, called neurons, organized in layers. The neurons receive input signals, apply mathematical transformations, and produce output signals that are passed on to the next layer. The layers are stacked hierarchically, with each layer learning increasingly complex features or representations of the input data.

One of the key advantages of deep learning is its ability to handle and extract meaningful features from large-scale datasets. Deep neural networks can learn intricate representations of images, text, audio, and other forms of data, leading to breakthroughs in computer vision, natural language processing, predictive maintenance, and many other fields. Convolutional neural networks (CNNs) are widely used in image-related tasks, while recurrent neural networks (RNNs) are commonly employed for sequential and language-based data.

Deep learning has also benefited from advancements in hardware and computational resources, as training deep neural networks often requires significant computational power. Graphics processing units (GPUs) and specialized hardware accelerators, such as tensor processing units (TPUs), have enabled faster training and inference of deep learning models.

In conclusion, deep learning is a powerful branch of machine learning that uses deep neural networks to learn hierarchical representations from data. Its ability to automatically learn features from raw data has revolutionized numerous fields and led to breakthroughs in various applications. As hardware and algorithms continue to advance, DL is poised to drive further innovation and impact a wide range of industries, shaping the future of artificial intelligence.

DL involves training artificial neural networks in order to detect patterns in large unstructured data sets. These ANNs, containing several hidden layers and performing complex tasks with minimal human interference, are mostly called Deep Neural Networks (DNNs).

I.3.3.1 Deep Neural Networks

A Deep Neural Network (DNN) is simply an artificial neural network containing a large number of hidden layers, which explains the term "deep".

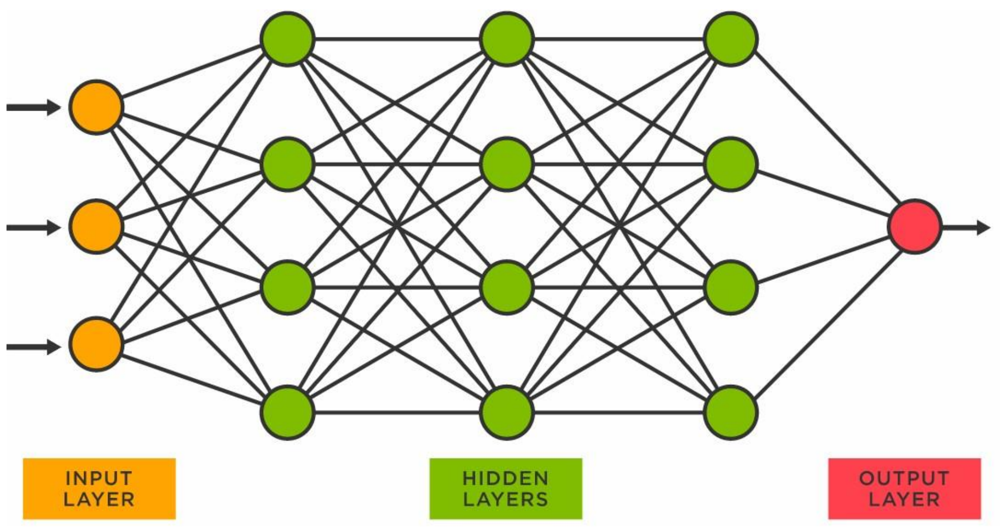


Figure I.5 Structure of a deep neural network

Deep neural networks typically consist of more than one hidden layer, organized in deeply nested network architectures. Furthermore, they usually contain advanced neurons in contrast to simple ANNs.

Therefore, DNNs may use multiple advanced operations in one neuron rather than using a simple activation function. These characteristics allow deep neural networks to be fed with raw input data and automatically discover a representation or output that is needed for the corresponding learning task. This is the networks' core capability, which is commonly known as deep learning [8].

While there are numerous types of DNNs, the most widely known are Convolutional Neural Networks and Recurrent Neural Networks.

I.3.3.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a type of neural networks commonly used for image and video processing tasks, such as image classification, object detection, and image segmentation. CNNs consist of multiple layers of convolutional and pooling operations that learn to extract features from images. In other words, CNNs have the human-like ability to recognize and classify objects based on their appearance [9].

One of the significant advantages of CNNs is their ability to automatically learn hierarchical representations from raw image data. By employing multiple layers, CNNs can capture increasingly complex and abstract features, enabling them to perform tasks such as image classification, object detection, and semantic segmentation. CNN architectures, such as AlexNet, VGGNet, ResNet, and InceptionNet, have achieved remarkable performance in various computer vision benchmarks and competitions.

Based on the dimension of the training data, CNNs can be devised into 1D CNNs and 2D CNNs. Deep 2D CNNs with many hidden layers and millions of parameters have the ability to learn complex objects and patterns providing that they can be trained on a massive size visual database with ground-truth labels. With proper training, this unique ability makes them the primary tool for various engineering applications for 2D signals such as images and video frames.

Yet, this may not be a viable option in numerous applications over 1D signals especially when the training data is scarce or application specific. To address this issue, 1D CNNs have recently been proposed and immediately achieved state-of-the-art performance levels in several applications such as anomaly detection and identification in power electronics and electrical motor fault detection. Another major advantage is that a real-time and low-cost hardware implementation is feasible due to the simple and compact configuration of 1D CNNs that perform only 1D convolutions (scalar multiplications and additions) [10].

In summary, CNNs are a key architecture in deep learning, particularly for computer vision tasks. Their ability to automatically learn and extract features from images has led to significant advancements in various applications. By leveraging convolutional and pooling layers, CNNs can effectively capture spatial patterns and hierarchical representations, enabling them to achieve state-of-the-art performance in image recognition, object detection, and other computer vision tasks.

I.3.3.3 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a type of neural networks that excel in processing sequential data, such as time series, text, and speech. RNNs contain feedback connections that allow information to flow in both directions, unlike feedforward neural networks which only flow in one direction. The feedback connections in RNNs allow the network to process sequential data and capture temporal dependencies, by passing information from one step to the next. At each step, the current input is combined with the previous hidden state to produce a new hidden state and output. This process is repeated for each step in the sequence, allowing the network to capture the context and dependencies of the input data [12].

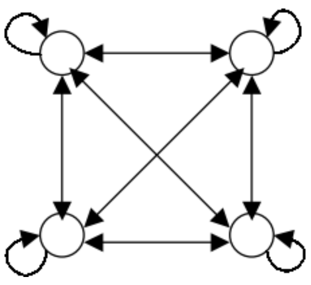


Figure I.6 An example of a fully connected RNN [13]

In the last years, several RNN architectures have been developed to meet the industries' standards, some of the most popular ones are the fully-connected RNN (FRNN), the long short-term memory (LSTM), and the gated recurrent unit (GRU).

- **FRNNs** connect the output of the previous time step with the additional input of the next time step, preserving important information about different time steps in the network [13].
- **LSTM** architecture has one cell state and three gates: an input gate, an output gate, and a forget gate. The cell state acts as a memory, while each gate functions like a conventional neuron, providing a weighted sum of its inputs. The forget gate decides what information to retain from previous steps. The input gate decides what information to add from the current step. The output gate decides what the next hidden state should be. Hence, only relevant information can pass through the hierarchy of the network. Thus, the LSTM has mechanisms to process both short-term and long-term memory components [14].

- **GRU** is similar to LSTM but has only two gates: an update gate and a reset gate. The update gate works similarly to the forget gate and the input gate of LSTM. It decides what information to throw away and what to add. The reset gate decides how much past information to forget. GRU has fewer parameters, uses less memory, and is faster to train than LSTM [14].

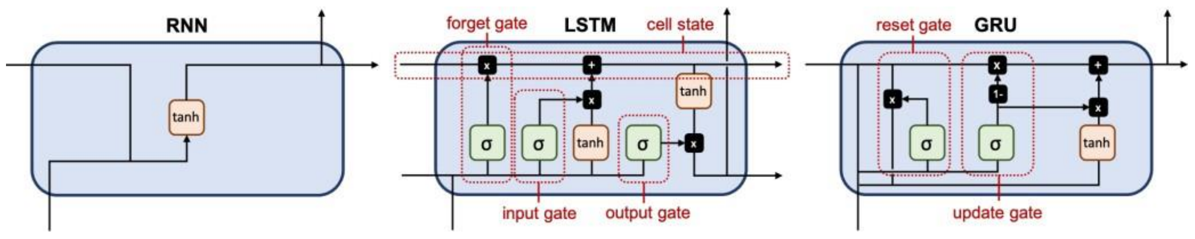


Figure I.7 Diagram of different recurrent units [14]

In conclusion, RNNs are powerful neural network architectures designed for processing sequential data. Their ability to capture dependencies across time steps enables them to model complex temporal patterns and make predictions or generate outputs based on sequential inputs. With variants like LSTM and GRU, RNNs have achieved state-of-the-art results in various sequential tasks, making them a fundamental tool in fields such as natural language processing, speech recognition, and time series analysis.

I.4 Problem statement

Predictive maintenance is a very advanced maintenance technique, it requires a large amount of data to properly function and predict future failures before occurring. In order to properly handle the large amount of data required, the implementation of AI and specifically DL models might be essential.

In our thesis, we chose to use and study the Long Short-Term Memory architecture LSTM as our main model for its ability to deal with large datasets and extract patterns to perform efficient training. LSTM networks can capture long-term dependencies and model contextual information making them particularly useful for tasks involving sequential data with temporal dynamics. Thus, LSTM is perfectly suited to treat, train and learn from our database which contain a large number of sequential vibration data.

To prove the LSTM efficiency, we will create a regular Autoencoder model using Python programming language and the TensorFlow machine learning framework, and compare its performance with our main LSTM-based Autoencoder model.

The two models will be trained on the exact same database, and evaluated on three primary points:

- Training time
- Loss function
- MSE anomalies

By comparing these two models, we will be able to see how the LSTM layers affect the performance of a regular Autoencoder, and make deductions from the results gathered.

I.5 Related work

Bughin, Jacques, et al. (2018). presented a discussion paper on how AI will impact the global economy in the future, the potential of AI for boosting the global economy was reviewed, and concluded that it will be impressive and visible over time [2].

Peres, Ricardo Silva, et al. (2020). published a paper pinpointing the current landscape of AI in manufacturing. A systematic review of different journals and science source materials was made to better understand the requirements and steps necessary for a successful transition into Industry 4.0 supported by AI and the challenges that may arise during this process [4].

Zhang, C., & Lu, Y. (2021) presented a paper that provides a state-of-the-art analysis of the ongoing and upcoming AI research. Noting that AI is a multidisciplinary field with various applications in numerous domains, concluding that the next advancement in this field can not only provide computers better logical thinking powers but can also give them emotional capabilities. It's possible that soon machine intelligence may surpass human intelligence [7].

Zonta, Tiago, et al. (2020) presented a survey that discusses the current obstacles and restrictions in PdM while also categorizing this field of study in regard to Industry 4.0's requirements. They concluded that computer science, including AI and distributed computing areas, is increasingly prominent in a field where engineering was predominant, highlighting the necessity of a multidisciplinary approach to properly meet Industry 4.0 [15].

Poór, P., Basl, J., & Zenisek, D. (2019) published an article to familiarize the reader with maintenance challenges in the industry. The historical overview of maintenance was mentioned. In the article, potential for a "new" kind of maintenance associated with Industry 4.0, namely PdM, was proposed. They concluded that PdM, being the most advanced form of all maintenance, is what companies strive to develop and what can give them an advantage over others [42].

Çınar, Zeki Murat, et al. (2020). provided a study about the recent advancements of ML techniques applied to PdM, the most commonly used ML algorithms for PdM were mentioned, and the continuous growth of PdM was highlighted [1].

Theissler, Andreas, et al. (2021). surveyed papers related to the automotive industry from an ML perspective, mentioning the adequacy of ML for PdM, concluding that the implementation of DL techniques will increase but requires the availability of large amounts of labelled data [6].

Serradilla, Oscar, et al. (2022). Published an article that aims at facilitating the task of choosing the right DL model for PdM, by reviewing cutting-edge DL architectures, and how they integrate with PdM to satisfy the needs of industrial companies (anomaly detection, root cause analysis, remaining useful life estimation). They are categorized in industrial applications, with an explanation of how to close any gaps. Open difficulties and potential directions for further research are then outlined [11].

Janiesch, et al. (2021). published an article summarizing the fundamentals of ML and DL to generate a broader understanding of the systematic framework of current intelligent systems. They abstractly defined keywords and concepts, described how to develop automated analytical models using ML and DL, and talked about the difficulties in applying such intelligent systems in the context of electronic marketplaces and networked commerce [8].

Zhou, Z. H. (2018). presented an article that reviews the state of supervised learning research, concentrating on three common forms of weak supervision: incomplete supervision, inexact supervision, and inaccurate supervision. It was determined that when there is a multitude of training instances with ground-truth labels, supervised learning techniques have had remarkable success. However, in practical applications, gathering supervision information incurs costs, making the ability to perform weakly supervised learning often beneficial. [47]

Van Engelen, J. E., & Hoos, H. H. (2020). presented a description of semi-supervised learning as a field. The survey provides an up-to-date analysis of this crucial area of ML, covering techniques from the early 2000s as well as more recent developments. Additionally, they have introduced a new taxonomy for semi-supervised categorization techniques that makes distinctions between the approach's main goal and how unlabeled data is employed [49].

Dike, Happiness Ugochi, et al. (2018). Published a paper that explores the training and learning of ANN-based unsupervised learning. It gives an explanation of the procedures for choosing and fixing a number of hidden nodes in an ANN-based unsupervised learning environment. Additionally, a summary of the status, advantages, and difficulties of unsupervised learning is provided. [48]

François-Lavet, Vincent, et al. (2018) provided a manuscript introducing deep RL models, algorithms, and techniques. Focusing in particular on the generalization features and the practical uses of deep RL [41].

Kiranyaz, Serkan, et al. (2021). wrote a paper that provides a thorough analysis of the fundamental design ideas and technical uses of 1D CNNs, with a particular emphasis on current advancements in this area. Finally, their distinctive qualities are highlighted, capping off their cutting-edge performance [10].

Salehinejad, Hojjat, et al. (2017). published a paper presenting a survey on RNNs and several new advances for newcomers and professionals in the field. The fundamentals and recent advances are explained and the research challenges are introduced, mentioning the LSTM and other RNNs architectures [12].

Na Pattalung, T., Ingviya, T., & Chaichulee, S. (2021). proposed a data-driven approach that combines RNNs with graspable explanations for predicting the probability of mortality. This method was able to identify and clarify the historical contributions of the linked elements to the prediction, in addition to providing the anticipated mortality risk. It was determined that if patients' clinical observations in the ICU are continually monitored in real time, they may benefit from early intervention [14].

I.6 Conclusion

With the advancement of technologies, and at the age of “Big-Data” where industries are generating and collecting vast amounts of data from a variety of sources at an unprecedented scale, predictive maintenance can be particularly effective at improving the reliability of the machinery and industrial processes in general.

The vast amounts of data available present both an advantage and a challenge to the predictive maintenance approach, because although having a lot of data can help PdM algorithms predict failures and identify patterns, properly using these amounts of data can be challenging, and one of the best ways to take on such a challenge is by using AI and deep learning algorithms, such as DNNs, CNNs, and RNNs.

Chapter II

Long Short-Term Memory Networks

Chapter II
Long Short-Term Memory Networks

II.1 Introduction

Predictive maintenance is a vital process that helps organizations optimize the maintenance of their machinery and equipment by predicting and preventing equipment failure. To achieve this, it is crucial to have accurate and timely predictions of equipment failures, which can be achieved through machine learning techniques such as RNN's Long Short-Term Memory (LSTM).

LSTM is a type of neural network that has been proven to be highly effective at modeling sequential data, making it well-suited for predictive maintenance tasks. By analyzing historical data on equipment performance, LSTMs can learn patterns and correlations that are indicative of future equipment failure.

In this Chapter, we are going to explain in detail what exactly are LSTMs and LSTM Autoencoders, including their architecture, training, and how they can be used in the industrial sector with the PdM approach.

II.2 Long Short-Term Memory Networks

The typical feature of the RNN architecture is a cyclic connection, which enables the RNN to possess the capacity to update the current state based on past states and current input data. These networks, consisting of standard recurrent cells, have had incredible success with numerous challenges. Unfortunately, when the gap between the relevant input data is large, the above RNNs are unable to connect the relevant information [17].

To handle the "long-term dependencies," Hochreiter and Schmidhuber [54] proposed the long short-term memory (LSTM) model.

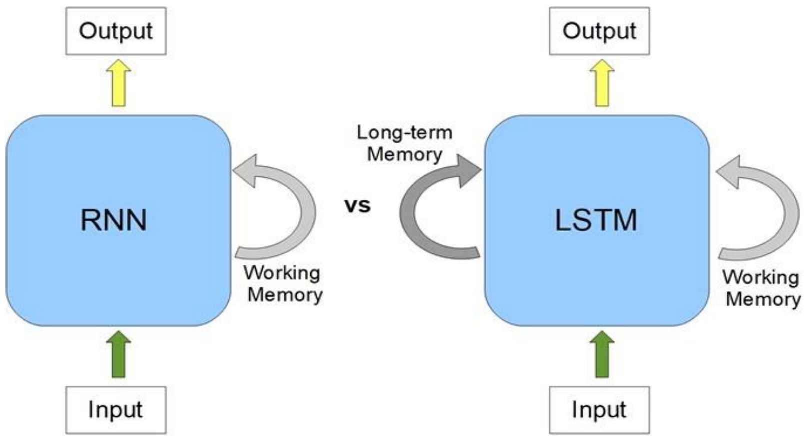


Figure II.1 RNN vs LSTM diagram [51]

Long short-term memory (LSTM) is a type of Recurrent Neural Network (RNN) that are particularly useful for working with sequential data, such as time series anomaly detection, which makes it convenient to implement in the PdM approach.

In particular, LSTMs excel at handling the complex and dynamic nature of equipment performance data, which often contains multiple variables and dependencies. By capturing the long-term dependencies in the data, LSTMs can provide more accurate predictions of future equipment failures, enabling organizations to take preventive measures before failures occur [18].

II.2.1 LSTM Architecture

The LSTM architecture consists of several units, each containing three main components: the input gate, the forget gate, and the output gate. These gates work together to control the flow of information into and out of the memory cell [19].

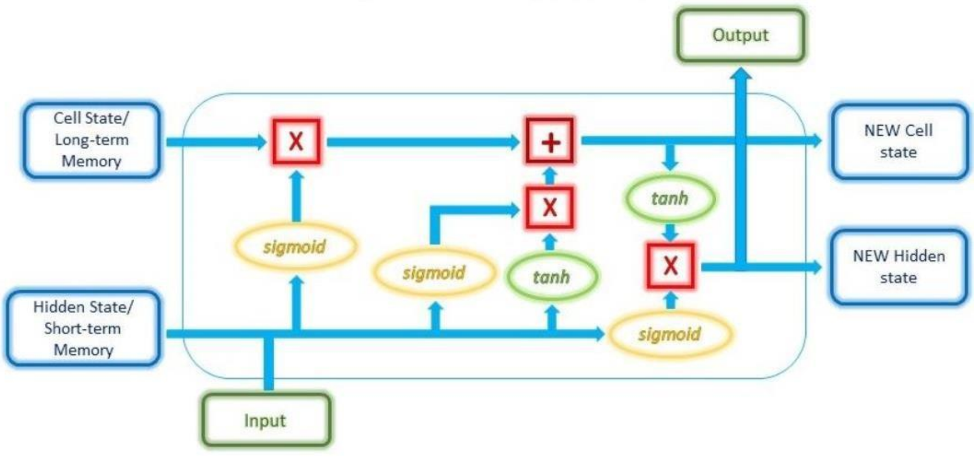


Figure II.2 LSTM gates diagram [52]

In the LSTM network and generally any neural network, a lot of computations happen to produce the appropriate output needed. Activation functions are necessary and play a vital role in these computations.

Activation functions are mathematical functions that introduce non-linearity to the output of a neural network's neurons or nodes. These functions are essential in neural networks because they allow the network to learn and model complex relationships between inputs and outputs. They introduce non-linearities, enabling the network to approximate non-linear functions, make complex decisions, and handle diverse data distributions.

There are several activation functions used in neural networks, the most common ones are the Sigmoid function, the Hyperbolic Tangent (Tanh) function, and the Rectified Linear Unit (ReLU) Function.

- **Sigmoid function:** The sigmoid function, also known as the logistic function, maps the input to a value between 0 and 1. It has an S-shaped curve and is often used in binary classification problems.
  The mathematical formula of the sigmoid function can be expressed as followed:

$$sigmoide(x) = \sigma(x) = {}^{1}\!/_{1 + \exp(-x)} \qquad \text{II.1}$$

- **Hyperbolic Tangent (Tanh) function:** The hyperbolic tangent function is similar to the sigmoid function but maps the input to a value between -1 and 1. It has a symmetric S-shaped curve and is useful in classification tasks. The mathematical formula of the Tanh function can be expressed as followed:

$$\tanh(x) = {}^{\exp(x) - \exp(-x)}\!/_{\exp(x) + \exp(-x)} \qquad \text{II.2}$$

- **Rectified Linear Unit (ReLU) Function:** The ReLU function is a popular choice in deep learning due to its simplicity and effectiveness. It outputs the input directly if it is positive, and 0 otherwise. The equation for the ReLU function is:

$$ReLU(x) = \max(0, x) \qquad \text{II.3}$$

II.2.1.1 The cell state and hidden state

The cell state is the long-term memory and the core of the LSTM network, its main role is to store information over long time periods and can be selectively read, written to, and erased by the network. It can be thought of as a conveyor belt that runs through the entire chain of LSTM cells, carrying information from the previous cell to the next one.

In each time step, the cell state is updated based on the three gates: the input gate, the forget gate, and the output gate. The input gate determines which information should be stored in the cell state, the forget gate determines which information should be erased, and the output gate determines which information should be used to compute the output [20].

On the other hand, the hidden state is the output of each cell in the LSTM network, which is a function of the current input and the previous hidden state. It is also sometimes referred to as the short-term memory of the network, as it stores information over short periods of time.

Unlike the cell state, the hidden state is not directly involved in the gating mechanisms of the LSTM, but it is computed from the cell state and can be thought of as a summary of the information stored in the cell state.

To summarize, the cell state and hidden state are both important components in LSTM networks, with distinct roles. The cell state stores information over long time periods and is updated based on the gating mechanisms of the LSTM. The hidden state, on the other hand, is the output of each cell and stores information over short periods. While they both carry information from the previous cell to the next one, the cell state is the primary carrier of long-term information, while the hidden state summarizes the information stored in the cell state and is used for making predictions or passing information to downstream tasks.

II.2.1.2 The input gate

The input gate determines which information is relevant to the current time step and should be stored in the memory cell. It takes as input the current input and the previous hidden state and applies an activation function (typically a sigmoid function) to each component. The sigmoid function is commonly used in neural networks as an activation function for binary classification problems [21].

In the input gate, the sigmoid function is used to "gate" the input by deciding which values should be allowed into the memory cell and which should be ignored, it outputs values between 0 and 1, which can be interpreted as the degree of importance that should be assigned to each input value. Values closer to 0 will be less important and may be ignored, while values closer to 1 will be considered important and allowed into the memory cell.

While using only the sigmoid function in the input gate of an LSTM may be sufficient in some cases and could simplify the model’s architecture, using a combination of sigmoid and hyperbolic tangent (tanh) function is the most common approach.

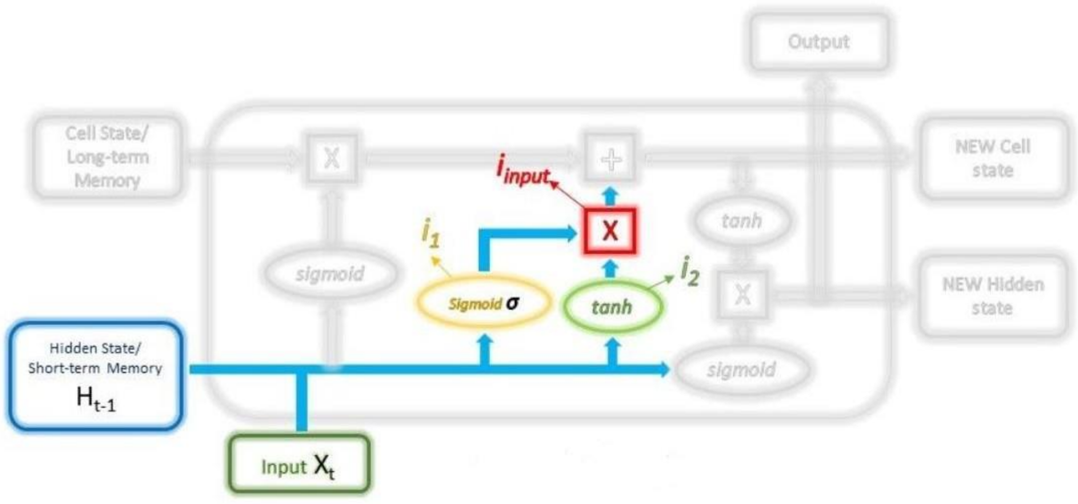


Figure II.3 Input gate computations diagram [52]

The sigmoid function alone is capable of regulating the flow of information, but it doesn't provide any control over the range of values that can be stored in the memory cell, that is why the tanh function is implemented.

The tanh function is another widely used activation function in neural networks. It maps the input values to a range between -1 and 1, which helps to control the range of the values that are stored in the memory cell. This is important because the memory cell can store both positive and negative values, and the tanh function helps to keep these values (called candidate values) stable by preventing them from growing too large or too small.

By combining these two activation functions, the input gate can effectively "gate" the input and store the relevant information in the memory cell, while filtering out irrelevant information and avoiding any possible instability in the learning process.

This combination can be achieved by using two layers, the sigmoid function and the hyperbolic tangent function would be applied respectively to the first and second layers.

As the first layer is being trained, the “weights” (parameters that are learned during the training process) in the sigmoid function will be updated such that it learns to filter the information received as previously explained.

The calculations of the first layer can be represented by the following equation:

$$i_1 = \sigma(W_{i_1} \cdot (H_{t-1}, x_t) + bias_{i_1}) \qquad \text{II.4}$$

Where:

- W_{i_1} is the weight matrix of the first layer i_1
- H_{t-1} is the previous hidden state

- x_t is the current input
- $bias_{i_1}$ is a vector added to improve the accuracy of the model

The second layer represents the calculation of the candidate values, regulating the network by passing the previous hidden state and current input into the hyperbolic tangent function, as followed:

$$i_2 = tanh(W_{i_2} \cdot (H_{t-1}, x_t) + bias_{i_2}) \qquad \text{II.5}$$

The outputs of these two layers are then multiplied and the information that needs to be stored in the memory cell results:

$$i_{input} = i_1 \cdot i_2 \qquad \text{II.6}$$

II.2.1.3 The forget gate

The forget gate in an LSTM network determines which information in the memory cell should be forgotten or discarded, based on the current input and the previous hidden state. Its main role is to prevent the network from remembering irrelevant or outdated information, which could lead to overfitting or poor performance [22].

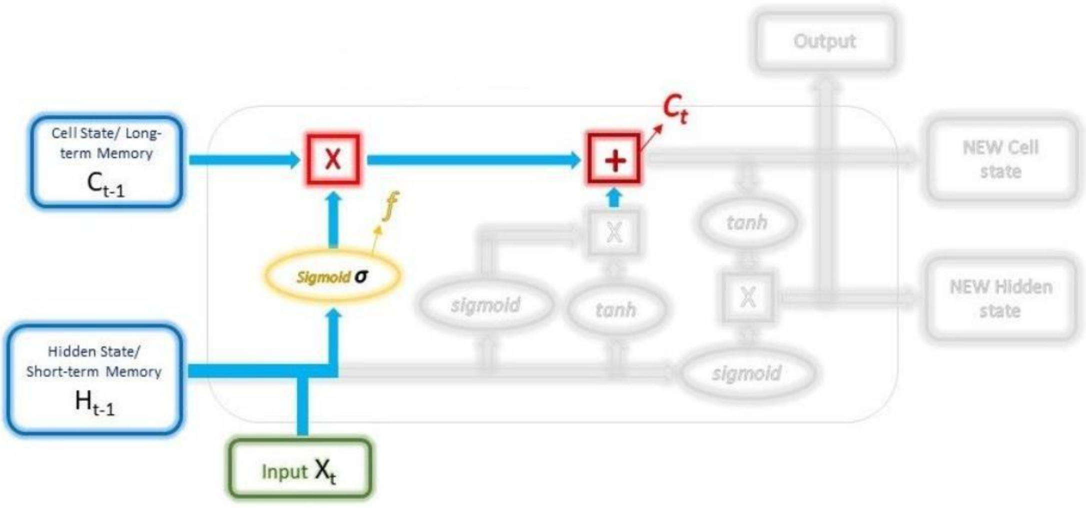


Figure II.4 Forget gate computations diagram [52]

To achieve this, the LSTM's forget gate calculates a forget vector, which is a set of values between 0 and 1 that determine how much of each element in the previous long-term memory should be preserved or forgotten. The forget vector is created by passing the concatenation of the current input and the previous short-term memory through a sigmoid activation function. This sigmoid function maps the input to a range between 0 and 1, similarly to the input gate, with values closer to 0 indicating that the corresponding element in the previous long-term memory should be forgotten, and values closer to 1 indicating that the element should be preserved.

The forget vector has values ranging from 0 to 1 and can be mathematically represented by the following equation:

$$f = \sigma(W_{forget} \cdot (H_{t-1}, x_t) + bias_{forget}) \qquad \text{II.7}$$

Once the forget vector is calculated, it is multiplied element-wise by the previous long-term memory to obtain the new long-term memory, as follows:

$$C_t = f \odot C_{t-1} \qquad \text{II.8}$$

Where:

- C_t is the new long-term memory
- f is the forget vector
- $\odot$ represents the element-wise multiplication
- C_{t-1} is the previous long-term memory

The new long-term memory is then updated with the information from the current input using the input gate, which determines which parts of the current input should be added to the long-term memory.

$$C_t = f \odot C_{t-1} + i_{input} \qquad \text{II.9}$$

This process effectively erases information from the previous long-term memory that is no longer relevant to the current input. By doing so, the network can learn to focus on the most important features of the input data and make better predictions or decisions.

II.2.1.4 The output gate

The output gate in an LSTM cell is a key component that determines which parts of the long-term memory and current input are passed on to the next cell or used as the final output of the network. It is responsible for regulating the flow of information and selectively passing on relevant information to subsequent time steps or as output [23].

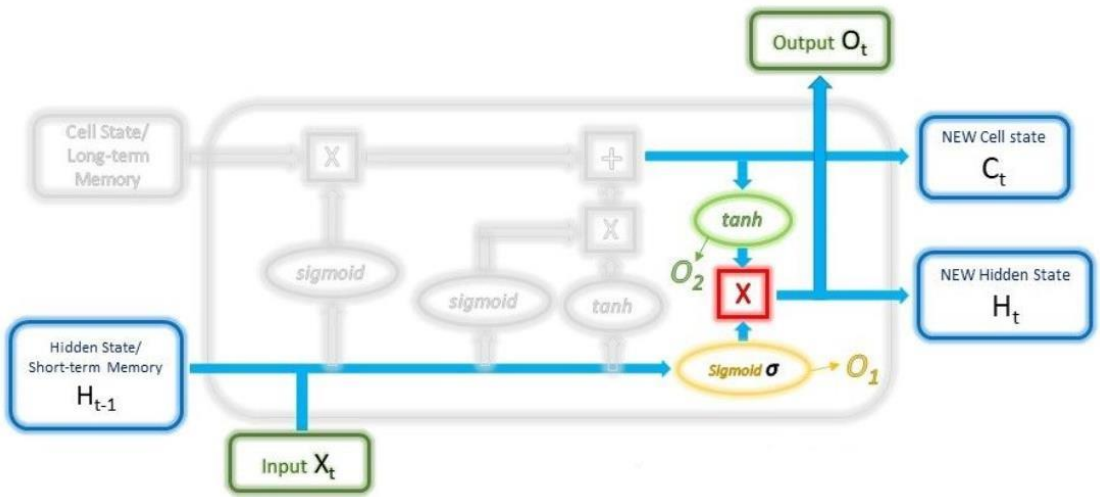


Figure II.5 Output gate computations diagram [52]

The output gate takes as input the current input, the previous hidden state, and the current long-term memory, which have all been processed by their respective gates (input and forget gates) as previously explained.

First, the current input and the previous hidden state are passed into the sigmoid activation function with the appropriate weights, which will determine the proportion of the current long-term memory that should be included in the new short-term memory.

$$O_1 = \sigma(W_{O_1} \cdot (H_{t-1}, x_t) + bias_{O_1}) \qquad \text{II.10}$$

Then, the tanh activation function is applied to the new long-term memory, which was calculated by the forget gate and updated by the input gate. This normalizes the values of the new long-term memory.

$$O_2 = tanh(W_{O_2} \cdot C_t + bias_{O_2}) \qquad \text{II.11}$$

The normalized new long-term memory is then multiplied element-wise with the output of the sigmoid gate to produce the new short-term memory:

$$H_t, O_t = O_1 \odot O_2 \qquad \text{II.12}$$

The hidden state/short-term memory and cell state/ long-term memory produced by these gates is then passed to the next time step for the process to be repeated or used as the final output of the network.

II.2.2 LSTM Applications

Long short-term memory networks have been widely used over the years and are constantly gaining popularity with the development of artificial intelligence and deep learning, therefore, LSTMs have a wide range of applications, such as Natural Language Processing (NLP), Image and Video Analysis, time series analysis and anomaly detection.

II.2.2.1 Natural Language Processing

Natural Language Processing (NLP) involves the use of computers to analyze, understand, and generate human language in various forms, including written text, spoken language, and even sign language.

LSTMs are widely used in NLP tasks such as language modeling, speech recognition, machine translation, and sentiment analysis, among others. One of the key advantages of LSTMs in NLP is their ability to handle variable-length input sequences and capture long-term dependencies in the data. This makes LSTMs particularly effective in tasks such as language modeling, where the model must predict the likelihood of a sequence of words based on the context of the previous words [24].

II.2.2.2 Image and Video Analysis

Image and video analysis is a field of study that involves the use of computers to interpret and understand visual information. This field has many applications, ranging from object recognition and image captioning to medical imaging and surveillance systems.

In the context of machine learning, image, and video analysis often involves the use of deep learning techniques, such as convolutional neural networks (CNNs) and LSTMs. While CNNs are particularly well-suited for image analysis tasks, such as identifying objects or classifying images into different categories, LSTMs have also shown great promise in the image and video analysis application, as they can be used to analyze the temporal aspect of video data, such as identifying changes in motion or tracking objects over time [25].

One of the main advantages of LSTMs in these applications is their ability to capture the temporal dependencies in the data and to maintain context over longer sequences. For example, in video analysis tasks, LSTMs can analyze a sequence of frames to identify objects or events that occur over time. In image captioning tasks, LSTMs can be used to generate descriptive captions that capture the content and context of the image.

II.2.2.3 Time Series Analysis

Time series analysis is a field of study that focuses on analyzing and modeling data that varies over time. This type of data is often collected at regular intervals, such as hourly, daily, or monthly, depending on the specific operation monitored, and can be found in many different domains, including finance, economics, and engineering [26]. Time series analysis involves identifying patterns and trends in the data, as well as forecasting future values, which could be very effective for applying predictive maintenance.

In recent years, there has been a growing interest in using machine learning techniques, such as deep learning, for time series analysis. Recurrent neural networks (RNNs), and LSTMs in particular, have emerged as a powerful tool for time series analysis and forecasting.

Time series data can exhibit complex temporal dependencies and nonlinear relationships that are difficult to capture with traditional statistical methods. LSTMs can learn these complex patterns by maintaining a memory of previous values, allowing them to effectively model long-term dependencies in the data [27].

One of the key advantages of LSTMs in time series analysis is their ability to handle variable-length sequences of data. This is particularly important for time series data, which can have varying lengths depending on the specific application. LSTMs are able to adapt to these variable-length sequences by using a gating mechanism that controls the flow of information through the network.

In addition to forecasting future values, LSTMs can also be used for anomaly detection, identifying unusual patterns or outliers in the data that may indicate equipment failure or other abnormal conditions.

II.2.2.4 Anomaly Detection

Anomaly detection is the process of identifying unusual or unexpected patterns or events in data. Anomalies can be caused by a variety of factors, including equipment failure, malicious activity, or changes in the underlying distribution of the data. Anomaly detection techniques are used in many different domains, including finance, healthcare, cybersecurity, and industrial automation [28].

One of the key challenges in anomaly detection is balancing the trade-off between sensitivity and specificity. A highly sensitive anomaly detection system will identify as many anomalies as possible, but may also generate a high number of false positives. On the other hand, a highly specific system may generate fewer false positives but miss some true anomalies. Balancing these trade-offs requires careful tuning of the anomaly detection algorithm and consideration of the specific domain and application.

In recent years, there has been growing interest in using deep learning techniques, such as LSTMs for anomaly detection.

In anomaly detection, the goal is to identify patterns in data that deviate significantly from what is considered normal or expected. Anomaly detection using LSTM networks is particularly effective for time series data, where patterns can change over time and may be difficult to detect using traditional methods [29].

To use LSTM anomaly detection, the first step is to train an LSTM network on normal data to learn the patterns and relationships in the time series. This training process involves feeding the LSTM network with historical data and optimizing the network's parameters to minimize the difference between the predicted and actual values. Once the network has been trained on normal data, it can be used to detect anomalies in new data.

When the LSTM network encounters a time series data point that deviates significantly from the learned patterns, it can flag that data as anomalous and alert users about potential issues. For example, in the context of predictive maintenance, an LSTM network trained on sensor data from industrial equipment can identify patterns that indicate potential equipment failures, allowing maintenance teams to take proactive measures to prevent downtime.

II.3 Autoencoders

Autoencoders are a type of neural networks that can learn to encode and decode different types of data. Commonly used in unsupervised learning tasks, the goal of an autoencoder is to learn a compressed representation of the input data in a lower-dimensional space, and then use this representation to reconstruct the original data as accurately as possible [30].

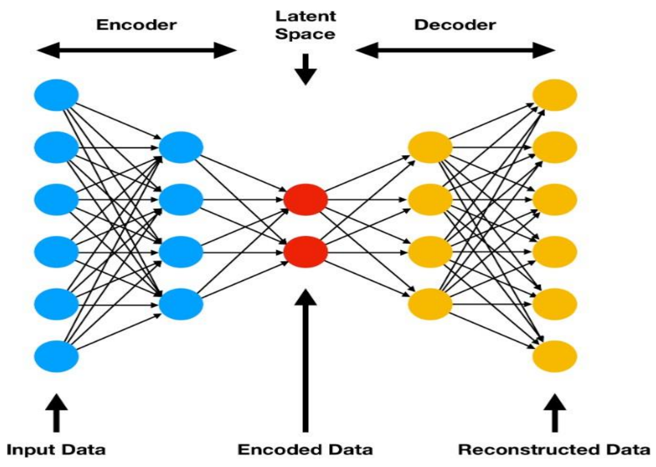


Figure II.6 Autoencoder model [53]

Autoencoders consist of two main parts: an encoder and a decoder. The encoder takes the input data and maps it to a lower-dimensional latent space, while the decoder takes the encoded data and reconstructs the original input data.

By training the network to minimize the difference between the input data and the reconstructed data, the autoencoder can learn to capture the most important features of the input data and ignore any irrelevant or noisy information.

Autoencoders have a wide range of applications, including data compression, image and speech recognition, and anomaly detection [30].

In anomaly detection, autoencoders can be used to identify patterns in temporal data by learning to encode the normal behavior of a system. The idea is to train the autoencoder on a dataset of normal, or non-anomalous, instances, and then use it to reconstruct new instances. When an anomalous instance is encountered, it will likely have a higher reconstruction error than normal instances, since it does not fit the learned pattern. Thus, the reconstruction error can be used as a metric for anomaly detection, and instances with high reconstruction error can be flagged for further investigation.

Autoencoders have several advantages over traditional anomaly detection methods, they can learn complex patterns in data and do not require explicit feature engineering. They are also able to adapt to new and changing patterns in the data, making them suitable for dynamic systems.

II.4 LSTM Autoencoders

Long Short-Term Memory autoencoders are a type of autoencoder that incorporates LSTM units in their architecture to handle sequential data.

Regular autoencoders, in their standard form, are not specifically designed to handle sequential data. They are primarily suited for static, non-sequential data such as images, tabular data, or fixed-length feature vectors, in contrast to LSTM autoencoders, they are typically not equipped to capture the temporal dependencies and sequential patterns present in sequential data.

LSTM autoencoders were created to address the unique challenges of modeling and reconstructing sequential data. By leveraging the temporal modeling capabilities of LSTM units, these models offer improved performance in capturing temporal dependencies, compressing sequential data, and generating/reconstructing sequences. They have become a valuable tool in various fields where sequential data analysis is required [31-32].

II.4.1 LSTM Autoencoders architecture

The LSTM autoencoder architecture combines the power of LSTM units for sequential modeling with the principles of autoencoders for learning compressed representations. This enables the model to effectively encode and reconstruct sequential data while preserving its essential characteristics.

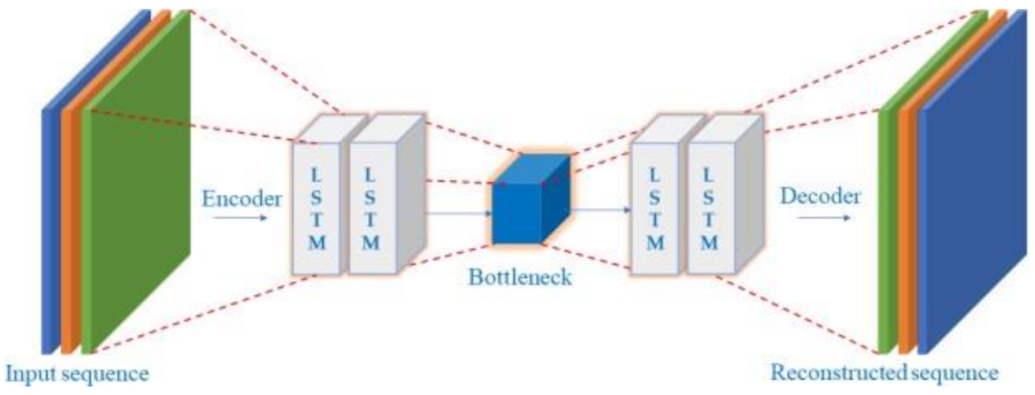


Figure II.7 LSTM Autoencoder model [33]

The architecture of an LSTM autoencoder consists of three main components: an LSTM encoder, a bottleneck layer (latent state), and an LSTM decoder [33].

II.4.1.1 LSTM Encoder

The LSTM encoder takes a sequential input and processes it step by step. At each time step, the LSTM unit computes the information detected (inputs) as described earlier. It receives an input and its hidden state from the previous time step, it then updates its hidden state and cell state based on the current input and the previous hidden state and cell state.

The encoder typically consists of multiple LSTM units stacked on top of each other, forming a deep LSTM architecture, this architecture processes the input sequence sequentially, capturing the temporal dynamics and dependencies within the data. It gradually encodes the sequential information into a compressed representation [34].

As the input sequence is processed step by step, the final LSTM layer in the encoder outputs the compressed representation or the latent space. This latent space represents a compressed and abstract representation of the input sequence, capturing its most salient features.

II.4.1.2 Latent space/bottleneck layer

The latent space or bottleneck layer in an LSTM autoencoder represents the compressed and abstract representation of the input sequence obtained from the encoder. It acts as an information bottleneck, capturing the essential features and patterns from the input sequence in a lower-dimensional representation. This layer effectively compresses the input data, reducing its dimensionality [35].

The term "bottleneck" highlights the idea that the representation space is narrowed down, analogous to a bottleneck in a physical sense where the flow of a substance is constrained to pass through a smaller opening. This analogy reflects the compression and dimensionality reduction that occur in the bottleneck layer of an LSTM autoencoder.

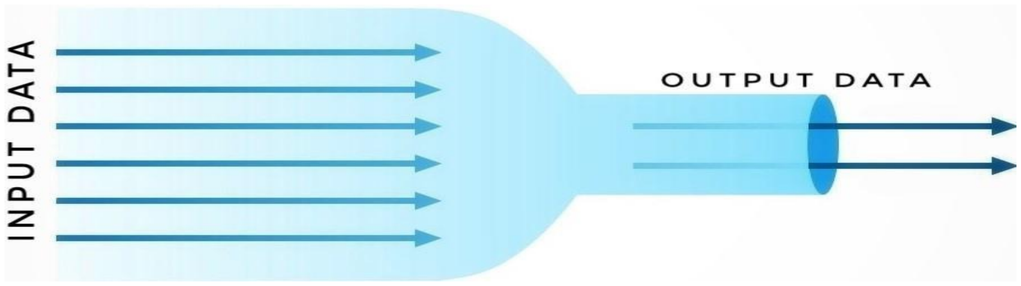


Figure II.8 Bottleneck representation

The dimensionality of the bottleneck layer is typically much smaller than the input sequence, representing a more compact and informative representation of the sequential data.

The bottleneck layer/latent space acts as the bridge between the encoder and the decoder, providing the encoded information, it serves as the foundation for the reconstruction process in the decoder.

II.4.1.3 LSTM Decoder

The decoder is responsible for reconstructing the original input sequence from the compressed representation obtained from the bottleneck layer. It plays a crucial role in the autoencoder's task of generating an output that closely resembles the input [35].

Similar to the encoder, the decoder typically consists of one or more LSTM layers. These layers receive the compressed representation, which serves as the initial input. The LSTM layers sequentially process this input, generating output at each time step.

At each time step, the LSTM layer takes the input from the previous time step and its own hidden state as inputs. It then performs computations to generate an output for the current time step. The hidden state of the LSTM layer captures the temporal dependencies and patterns in the data, allowing the decoder to generate a sequential output that reflects the original sequence.

By repeating this process for each time step, the decoder gradually reconstructs the original input sequence. Each LSTM layer builds upon the information from the previous time steps, utilizing its hidden state to inform the generation of the current output.

During training, the reconstruction loss is calculated to quantify the dissimilarity between the reconstructed output sequence and the original input sequence. A suitable loss function, such as mean squared error (MSE), is employed depending on the nature of the input data [36].

The goal of training the autoencoder is to minimize this reconstruction loss. By optimizing the model's parameters, the autoencoder learns to generate reconstructed output sequences that closely resemble the original input sequences. Minimizing the reconstruction loss encourages the autoencoder to capture and reproduce the most salient features of the input data.

The final output of the decoder is the reconstructed output sequence, which ideally should closely match the original input sequence. This reconstructed output can be used for various purposes, such as data analysis, anomaly detection, or generating predictions.

II.4.2 LSTM Autoencoders applications

LSTM autoencoders have emerged as powerful tools with a wide range of applications in the field of sequence modeling and analysis. Their ability to capture complex temporal dependencies, compress input data into a latent space, and reconstruct the original sequence make them well-suited for various tasks.

From anomaly detection and data compression to time series forecasting, LSTM autoencoders offer versatile solutions in different domains such as cybersecurity, finance, and manufacturing. By leveraging the strengths of LSTM networks in combination with the reconstruction capabilities of autoencoders, these models have proven effective in addressing real-world challenges and extracting valuable insights from sequential data.

II.4.2.1 Time Series Forecasting

Time series forecasting is one of the most common and well-established applications of LSTM autoencoders. The particular ability of LSTM autoencoders to capture temporal dependencies and patterns in sequential data, make them highly effective for time series forecasting tasks.

In time series forecasting, the goal is to predict future values or patterns based on past observations. LSTM autoencoders can learn to encode the input time series into a compressed representation or latent space, capturing the most relevant information. The decoder part of the autoencoder can then generate future predictions based on this compressed representation.

While regular LSTM networks can also be used for time series forecasting, LSTM autoencoders can offer additional benefits in this area due to their enhanced ability to capture temporal dependencies and model complex patterns.

The unsupervised nature of autoencoders allows them to learn from unlabeled data without explicit future targets, making them adaptable to scenarios where future labels are scarce or unavailable. Furthermore, the compression aspect of autoencoders enables them to extract and represent the most relevant features from the input time series, reducing the dimensionality and focusing on key information for accurate forecasting [37].

By leveraging the reconstruction loss during training, LSTM autoencoders learn to capture the intricate temporal dependencies in the data, making them highly effective in capturing long-term patterns and handling irregularities. These characteristics, combined with the ability to detect anomalies through higher reconstruction errors, make LSTM autoencoders well-suited for time series forecasting, providing a powerful tool to model and predict complex temporal dynamics in various domains.

II.4.2.2 Data Compression

Data compression refers to the process of reducing the size of data files or streams without losing essential information. The goal of data compression is to minimize storage space requirements, reduce transmission bandwidth, and improve overall efficiency in handling and processing data.

While regular LSTM networks can perform data compression to some extent, LSTM autoencoders are generally better suited for data compression tasks.

LSTM autoencoders are specifically designed with a bottleneck layer that serves the purpose of compressing the input data into a lower-dimensional latent space. The training objective of the autoencoder is to reconstruct the original input sequence from this compressed representation. By leveraging this reconstruction process, the autoencoder learns to capture and encode the most salient features of the input data, effectively reducing its dimensionality.

Regular LSTM networks, on the other hand, are primarily designed for sequence modeling and prediction. While they can implicitly capture temporal dependencies and represent sequential patterns, their primary focus is not on explicit data compression.

LSTM autoencoders, with their dedicated architecture for compression, tend to offer better compression capabilities compared to regular LSTM networks. The inclusion of the bottleneck layer in LSTM autoencoders specifically facilitates the extraction and representation of essential features in a lower-dimensional space [35, 36, 37].

Therefore, when it comes to data compression tasks, LSTM autoencoders are generally preferred over regular LSTM networks due to their specialized design and the explicit inclusion of a compression mechanism in their architecture.

II.4.2.3 Anomaly Detection

As previously explained, anomaly detection is the process of identifying patterns or instances that deviate significantly from the norm or expected behavior within a dataset. An anomaly refers to a data point or a set of data points that do not conform to the usual patterns or behaviors observed in the majority of the data.

Regular LSTM networks have been successfully applied to anomaly detection tasks and can effectively identify deviations from expected behavior. They are capable of capturing temporal dependencies and modeling sequential patterns, which can be advantageous for detecting anomalies in time series data.

On the other hand, LSTM autoencoders offer additional features that can be beneficial for anomaly detection. The unsupervised nature of autoencoders allows them to learn representations of normal data without requiring explicit anomaly labels. The reconstruction-based approach, where the model aims to reconstruct the input data, can help identify anomalies as instances with high reconstruction errors. The dimensionality reduction aspect of autoencoders can also enhance their ability to capture salient features and reduce the impact of noise [38].

However, it is important to note that the performance of both LSTM autoencoders and regular LSTM networks in anomaly detection tasks can vary depending on factors such as the complexity and nature of the data, the choice of hyperparameters, and the specific characteristics of the anomalies being targeted [39]. Therefore, it is recommended to conduct empirical evaluations and comparisons to determine which approach is more suitable for a particular anomaly detection scenario.

II.5 LSTM Autoencoders vs Regular LSTMs

Regular LSTMs and LSTM autoencoders are both deep learning models that are increasingly applied to various domains due to their ability to capture long-term dependencies and learn complex patterns in sequential data.

These two models present various similarities and also differences that make one of them more appropriate to use than the other depending on the task at hand.

II.5.1 Similarities

The are several similarities between the LSTM autoencoders and the regular LSTMs, these similarities can be resumed in the following points:

- **Architecture:** Both regular LSTMs and LSTM autoencoders are built upon the architecture of Long Short-Term Memory (LSTM) networks. They both utilize the recurrent nature of LSTMs to capture sequential dependencies and handle time series data.
- **Temporal Modeling:** Both models excel in modeling and analyzing temporal sequences. They are designed to handle time series data, where the order of data points matters, and capturing sequential patterns is crucial.
- **Flexibility in Input Data:** Both models can handle various types of input data, including numerical, categorical, or textual data. They can be adapted and configured to process different data modalities, making them versatile for a wide range of applications.

II.5.2 Differences

While the LSTM autoencoders have similarities with the regular LSTMs, they also present differences that make them stand off from each other, these differences can be expressed in the following points:

- **Objective and Task:** Regular LSTMs are primarily used for tasks such as sequence prediction, natural language processing, and sentiment analysis. They aim to model and predict the next element in a sequence. In contrast, LSTM autoencoders are designed for unsupervised learning tasks, such as data compression and anomaly detection.
- **Output and Reconstruction:** Regular LSTMs typically produce an output sequence, which is often the predicted value or label for each element in the input sequence. LSTM

autoencoders, on the other hand, focus on reconstructing the input sequence using the compressed latent space representation. The reconstruction objective guides the learning process and enables tasks like data compression and anomaly detection.

- **Training Data:** Regular LSTMs are often trained using labeled data, where each input sequence is associated with a corresponding target or label. LSTM autoencoders, on the other hand, can be trained with either labeled or unlabeled data. In the case of anomaly detection, they can be trained on normal data only and identify anomalies as instances with high reconstruction errors.
- **Latent Space Representation:** One key distinction of LSTM autoencoders is the presence of a latent space or bottleneck layer. This compressed representation captures the most salient features of the input sequence. Regular LSTMs do not have an explicit mechanism for dimensionality reduction or latent space representation.

II.6 Predictive Maintenance and Anomaly Detection

As discussed in Chapter I, predictive maintenance aims to predict and prevent equipment failures or malfunctions by monitoring and analyzing the condition of the equipment. Anomaly detection plays a crucial role in PdM as it helps identify abnormal patterns or behavior in the data collected from the equipment. It helps prevent equipment failures, improve maintenance efficiency, and enhance overall operational reliability [40].

By utilizing anomaly detection techniques, such as LSTM autoencoders, abnormalities or deviations from the normal operating behavior of the equipment can be detected. This can include unusual sensor readings, irregular patterns, or unexpected changes in the data. Identifying these anomalies early on can provide valuable insights into potential equipment failures or maintenance needs.

Predictive maintenance systems can utilize anomaly detection to trigger alerts or generate maintenance schedules based on the detected anomalies. By addressing potential issues proactively, organizations can minimize equipment downtime, reduce repair costs, and optimize maintenance operations.

II.7 The state of the art

Yu, Yong, et al. (2019) mentioned in this paper that LSTM has become the focus of DL. To investigate its learning capacity, they examined the LSTM cell and its variations. They also divided LSTM networks into two primary types: LSTM-dominated networks, and integrated

LSTM networks, and discussed their different applications. Finally, LSTM network research directions were outlined [17].

Li, Zhuohan, et al. (2018). designed an original LSTM training algorithm by leveraging the information flow. The training algorithm proposed can force the input and forget gates' values to 0 or 1, creating a robust LSTM model. The usefulness of the suggested training algorithm was demonstrated in experiments on language modeling and machine translation. [22].

Smagulova, K., & James, A. P. (2019). In this paper, they examined the starting point and motivations for the creation of the LSTM neural network, and offered an overview of the current LSTM approaches, highlighting the most recent advancements in memristive LSTM structures [21].

Okut, H. (2021). reviewed the training process of RNNs, and explained how the LSTM neural networks can handle the main weakness of RNNs by learning long-term dependencies [20].

BERRAJAA, A. (2022). In this paper, an RNN-LSTM sentiment analysis model was put forth. In order to provide structured knowledge that can be applied to certain tasks, the goal was to build systems capable of extracting subjective information from natural language documents, such as feelings and opinions. With a 96% success rate, the LSTM model's performance was quite remarkable [24].

Zhao, T. (2019, July). In this article, in order to categorize massive amounts of video data, they suggest certain RNN variants, such as stacked bidirectional LSTM/GRU networks with attention mechanisms. The model, which incorporates audio and visual data, produced excellent results. It was referred to this approach as Deep Multimodal Learning (DML) due to its multimodal features. This DML-based model was assessed in a well-known video classification competition on Kaggle hosted by Google [25].

Lindemann, Benjamin, et al. (2021). It was shown in this paper that stacked LSTM networks can learn higher-level temporal patterns without prior knowledge of the pattern duration and that it may be a practical method to model typical time series behavior, which can be used to detect anomalies [28].

Bank, D., Koenigstein, N., & Giryes, R. (2020). In this paper, Autoencoders, a specific type of neural network was introduced. The autoencoders' architecture, goal, and different applications were mentioned [30].

Saumya, S., & Singh, J. P. (2022). In this study, to identify between spam reviews and legitimate reviews, an unsupervised learning model integrating LSTM networks and autoencoder (LSTM-autoencoder) was suggested. The model in question was trained on how to identify real review trends from textual details only. The experimental findings demonstrate that the model can distinguish between legitimate and spam reviews with reasonable accuracy [31].

Kang, Jaeyong, et al. (2021). In this article, they propose a method for detecting anomalies using a one-class LSTM autoencoder. Only normal subsequences are used as training data for the model. The mean absolute error (MAE) for each subsequence is determined in order to identify anomalies in test data. An example is labeled an anomaly if the error exceeds a predetermined threshold that was set to the maximum value of MAE in the training (normal) dataset. The experiments used data from metro trains in Korea and showed good results [32].

Do, J. S., Kareem, A. B., & Hur, J. W. (2023). In this paper, an LSTM-autoencoder model was utilized for training and testing to improve the accuracy of the anomaly detection procedure. This strategy enabled identifying patterns and trends in the vibration data that might not have been obvious when using more conventional techniques. The accuracy percentage for finding anomalies in the vertical carousel system using the correlation coefficient model and LSTM-autoencoder was 97% [33].

Nguyen, H. Du, et al. (2021). This study proposed an LSTM network-based approach for multivariate time series data forecasting, in addition to an LSTM Autoencoder network-based approach coupled with a one-class SVM algorithm for anomaly detection in sales. The acquired results demonstrate that, in comparison to the LSTM-based method proposed in prior work, the LSTM Autoencoder-based method leads to improved performance for anomaly identification [36].

Bampoula, Xanthi, et al. (2021). This study addresses a strategy to facilitate the transition from a PM to a PdM approach. A DL algorithm is employed to enable such transition. To train and test a prototype implementation of LSTM-autoencoders for determining the remaining usable life of the monitored equipment, real-world data gathered from manufacturing operations is employed. Finally, a use case involving a manufacturing process for the steel sector is used to evaluate the proposed approach [39].

Kamat, P., & Sugandhi, R. (2020). The core of PdM, according to this research, is anomaly detection, with a primary goal of identifying anomalies in operational equipment at an early stage. The difficulties with conventional anomaly detection methods are discussed, and a unique DL method for predicting anomalies before actual machinery breakdown is suggested. The suggested system uses an unsupervised learning technique called autoencoders, a kind of deep learning that can be used to find new classes of anomalies [40].

II.8 Conclusion

In conclusion, this chapter delved into the concepts, architecture, and applications of LSTMs and LSTM autoencoders, highlighting the anomaly detection method used in our main predictive maintenance model and its advantages.

LSTMs are powerful recurrent neural networks that excel in modeling sequential data, capturing long-term dependencies, and making accurate predictions. They have found success in various domains such as language modeling and time series forecasting. On the other hand, LSTM autoencoders extend the capabilities of LSTMs by incorporating the benefits of autoencoders. They offer unique advantages in unsupervised learning tasks, data compression, and anomaly detection.

Anomaly detection is a well-suited technique for predictive maintenance approaches, by enabling early detection of abnormal behavior or patterns in equipment data, it plays a vital role in PdM. It helps prevent equipment failures, improve maintenance efficiency, and enhance overall operational reliability.

Chapter III

Autoencoder model

Chapter III
Autoencoder model

III.1 Introduction

After introducing predictive maintenance and how it can affect the reliability of numerous systems and machinery in general, and after presenting different deep learning approaches, mainly Autoencoders, LSTM Autoencoders, and how they are used for anomaly detection. It is time to introduce our first model.

In this chapter, we introduce our Autoencoder-based model, along with the system and database used to evaluate it. We are going to explain in detail every part of our program, point out the results gathered, and clarify these results using plots.

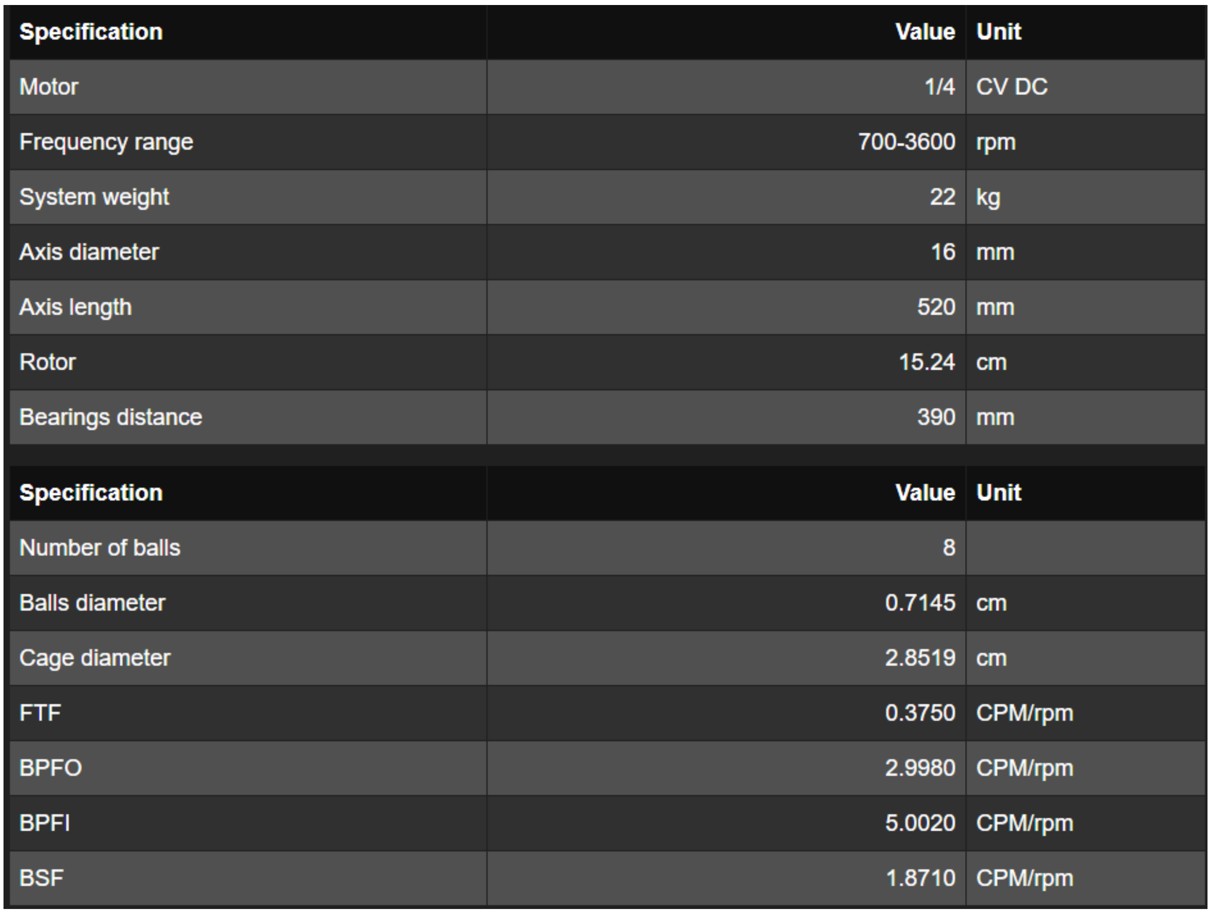
III.2 System & Database description

Before we introduce our program and model, we need to clarify the source and characteristics of our data and system. The database we used is composed of time-series data collected from sensors installed on a SpectraQuest's Machinery Fault Simulator (MFS) Alignment-Balance-Vibration (ABVT) system.

SpectraQuest is a company that specializes in providing solutions for machinery fault diagnosis, condition monitoring, and vibration analysis. They offer a range of products and services aimed at helping industries ensure the reliability, performance, and safety of their machinery and equipment.

The SpectraQuest's MFS ABVT is a specialized equipment designed to simulate various fault conditions and performance scenarios in machinery. It is commonly used for research, testing, and training purposes in the field of fault diagnosis and condition monitoring.

The following figure represent the characteristics of the MFS ABVT we used:



Specification	Value	Unit
Motor	1/4	CV DC
Frequency range	700-3600	rpm
System weight	22	kg
Axis diameter	16	mm
Axis length	520	mm
Rotor	15.24	cm
Bearings distance	390	mm

Specification	Value	Unit
Number of balls	8	
Balls diameter	0.7145	cm
Cage diameter	2.8519	cm
FTF	0.3750	CPM/rpm
BPFO	2.9980	CPM/rpm
BPFI	5.0020	CPM/rpm
BSF	1.8710	CPM/rpm

Figure III.1 Specifications of the MFS ABVT [55]

To collect the data, four sensors were used:

- Three **Industrial IMI Sensors, Model 601A01** accelerometers on the radial, axial and tangential directions:
  - Sensibility (±20%) 100 mV per g (10.2 mV per m/s^2).
  - Frequency range (±3 dB) 16-600000 CPM (0.27-10.000 Hz).
  - Measurement range ±50 g (±490 m/s^2).

- One **IMI Sensors triaxial accelerometer, Model 604B31**, returning data over the radial, axial and tangential directions:
  - Sensibility (±20%) 100 mV per g (10.2 mV per m/s^2).
  - Frequency range (±3 dB) 30-300000 CPM (0.5-5.000 Hz).
  - Measurement range ±50 g (±490 m/s^2).

Our database contains two simulated states:

- Normal functioning state: This state represents the normal operating condition of the machinery, where all components are functioning properly, and there are no faults or abnormalities.
- Imbalance state: This state simulates an imbalance in the rotating components of the machinery by adding weights ranging from 6g to 35g. Imbalance can occur due to uneven distribution of mass, leading to vibrations and performance issues.

In the dataset, there are 49 normal sequences without any faults. Each normal sequence corresponds to a fixed rotation speed ranging from 737 rpm to 3686 rpm, with an increment of approximately 60 rpm between each sequence.

For the imbalance sequences, the same 49 rotation frequencies used in the normal operation case are employed for loads below 30g. However, for loads equal to or above 30g, the resulting vibrations make it impractical for the system to achieve rotation frequencies above 3300 rpm. This limitation reduces the number of distinct rotation frequencies and measurements available.

To conclude, we used in our program a simulated database obtained from SpectraQuest's Machinery Fault Simulator, and after clarifying the system and database, we can now introduce our first program.

III.3 Autoencoder-based program

Our program consists of several Python commands, and we will explain every component of it. We divided the code into 4 main points:

- The libraries
- Data preparation
- Data preprocessing
- Autoencoder model

III.3.1 The Libraries

In the field of coding, a library refers to a collection of precompiled code and resources that provide specific functionality, features, or tools for developers. Libraries are designed to be reused, making development more efficient and enabling programmers to leverage existing code rather than building everything from scratch.

There are several libraries used in our program, some of which are part of the Python standard library, and some are not, meaning that they need to be preinstalled.

The libraries and modules used are:

- **glob:** The glob library provides a function that helps in finding files and directories whose names match specific patterns. It is useful for working with file paths and retrieving a list of files based on specific criteria.
- **Pandas:** The pandas library is a popular data manipulation and analysis tool. It provides data structures like DataFrames for efficient handling of structured data, as well as functions for data cleaning, filtering, merging, and more.
- **NumPy:** The NumPy library is a fundamental package for scientific computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently.
- **SkLearn:** Also known as Scikit-Learn, it is a comprehensive machine learning library in Python. It provides a wide range of algorithms and tools for tasks such as classification, dimensionality reduction, and model selection.
- **pyplot:** it is a module within the Matplotlib library that provides a MATLAB-like interface for creating and manipulating plots.
- **TensorFlow:** It is a library commonly used for deploying machine learning models and various DL.
- **Keras:** Keras is a powerful library that facilitates the development of DL models by providing a high-level and intuitive interface. It is widely used in the deep learning community and has become a popular choice for building neural networks due to its simplicity and flexibility. With its different modules, this library may be the most important one in our program, as it allows us to implement our model with relative ease, without directly having to program it.
- **random:** The **random** module allows you to generate random numbers, shuffle sequences, select random elements, and perform other randomization-related tasks.
- **time:** the **time** library in Python provides various functions and classes for working with time-related operations and measurements. It is a standard library in Python and does not require any external installation.

All these mentioned libraries and modules will be necessary for the right functioning of our program, the following lines of code show how the libraries are imported using the command "import":

```
import glob
import pandas as pd
import numpy as np
import sklearn
import matplotlib.pyplot as plt

import tensorflow as tf

import time
from keras.callbacks import Callback
from keras.layers import Dense, LSTM, RepeatVector, TimeDistributed,
Dropout, Input
from keras.models import Model
from keras import regularizers
from keras.callbacks import EarlyStopping

import random
```

After importing all the needed libraries and modules, we can begin the data preparation step.

III.3.2 Data preparation

In this step, we are going to prepare the data by downloading it to our program, clarify it using plots, and categorize it as needed.

III.3.2.1 Data download

```
cur_path = "/content/fault-induction-motor-dataset/imbalance"

normal_file_names = glob.glob("/content/fault-induction-motor-
dataset/normal"+'/normal/*.csv')

imnormal_file_names_20g = glob.glob(cur_path+'/imbalance/20g/*.csv')
```

These lines of code use the **glob.glob()** function to retrieve the file paths of the data used in our program. The second and third line retrieves the file paths for the normal functioning and the **20g** imbalance respectively.

After that, we define a function that will allow us to read the data as followed:

```
def dataReader(paths):
  data = pd.DataFrame()
  for i in paths:
    lowdata = pd.read_csv(i,header=None)
    data = pd.concat([data,lowdata],ignore_index=True)
  return data
```

The function takes a parameter "**paths**", which represent the file paths of our data. Inside the function, an empty data-frame named "**data**" is created using "**pd.DataFrame()**". This data-frame will be used to store the data from all the CSV files.

The function then iterates over each file path in the "paths" list using a for loop. The read CSV data is stored in a temporary data-frame named "**lowdata**". Then, the "**pd.concat()**" function is used to concatenate the "**lowdata**" with the existing "**data**". The "**ignore_index=True**" argument ensures that the concatenated data-frame has a continuous index.

Finally, the function returns the concatenated "**data**", which contains the combined data from all the CSV files.

To resume, the "**dataReader**" function will allow us to pass a list of file paths to it, and it will return a single data-frame containing readable data from all the data files.

We use the "**dataReader**" function to assign the normal functioning data and imbalance data to the variables "**data_n**" and "**data_20g**" respectively.

```
data_n = dataReader(normal_file_names)

data_20g = dataReader(imnormal_file_names_20g)
```

After that, we define another function named "**ref_data**" that will allow us to easily extract specific columns from the input data based on predefined indices.

```
def ref_data(data):

  # Convert the input data to a pandas DataFrame
  data = pd.DataFrame(data)

  # Check the number of columns in the data
  num_columns = data.shape[1]

  # Define the selected indices
  selected_indices = [1, 2, 3]

  # Validate the selected indices
  valid_indices = [i for i in selected_indices if i < num_columns]

  # Select the columns with the valid indices
  selected_data = data.iloc[:, valid_indices]

  # Return the selected data
  return selected_data
```

We use this function to extract the axial, radial, and tangential data columns from the normal functioning and the imbalance data as following:

```
data_n = ref_data(data_n)

data_20g = ref_data(data_20g)
```

III.3.2.2 Data visualization

To clarify the data used we will plot the three columns of both the normal and imbalance data as followed:

```
data_n.columns = ['axial', 'radial', 'tangential']

data_20g.columns = ['axial', 'radial', 'tangential']

fig, axs = plt.subplots(3, sharex=False, sharey=False, figsize=(15,
15))

for i, column in enumerate(data_n.columns):
    axs[i].plot(data_n[column][0:12000000:10000])
    axs[i].set_ylabel('m/s^2')
    axs[i].set_xlabel('number of samples')
    axs[i].set_title('{} vibration'.format(column))

plt.subplots_adjust(hspace=0.5)
plt.show()
```

These lines plot the axial, radial, and tangential data for the normal state, three plots will result after executing:

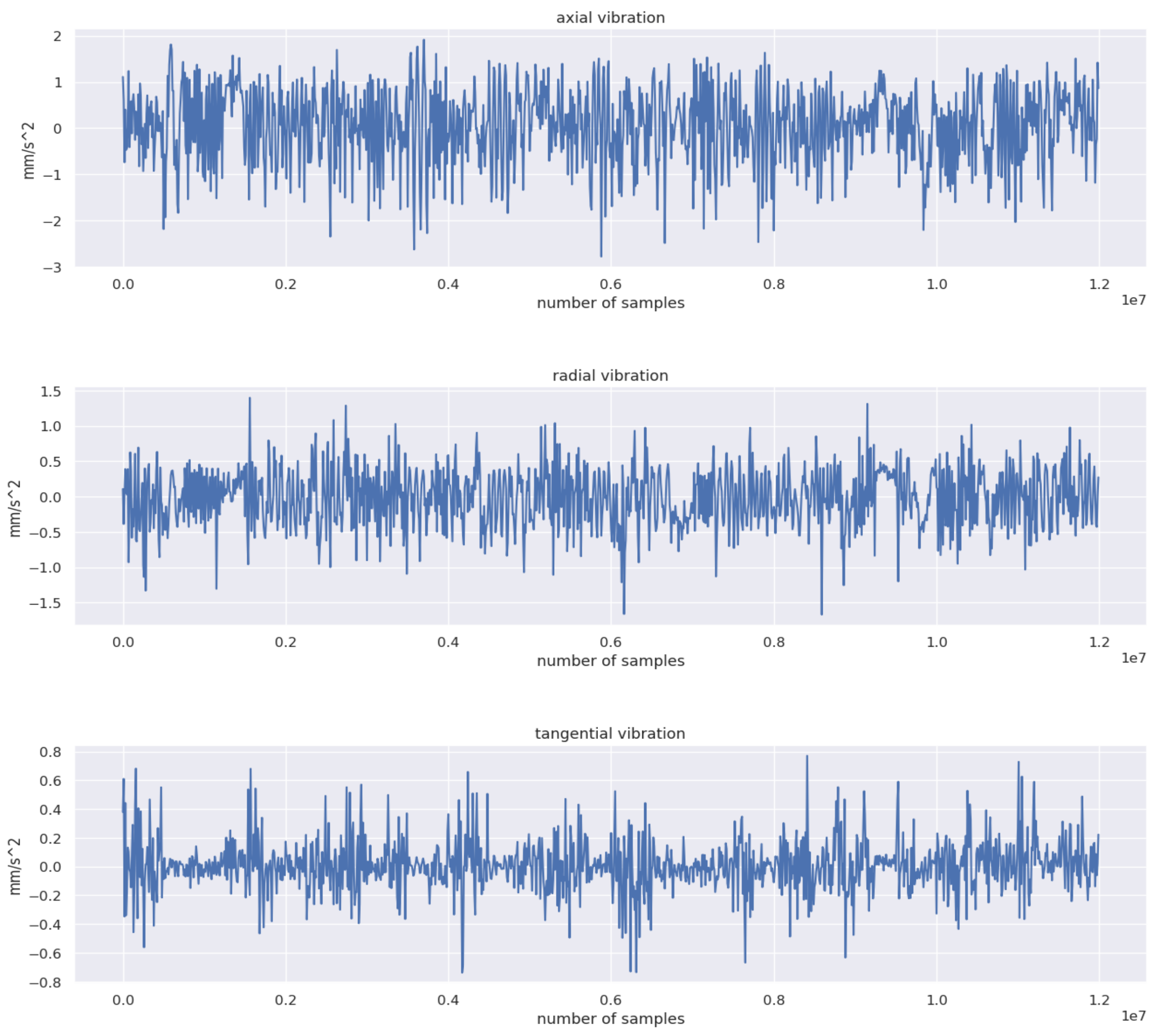


Figure III.2 Vibrations of normal state

To compare with the vibrations of the normal state, we choose to plot the imbalance data of the lightest, heaviest and average weights.

Using the same lines of code, by modifying only the input of the **enumerate** parameter, we plot the three axes of the 6g, 20g, and 35g imbalance data.

The results are showed in the following figures:

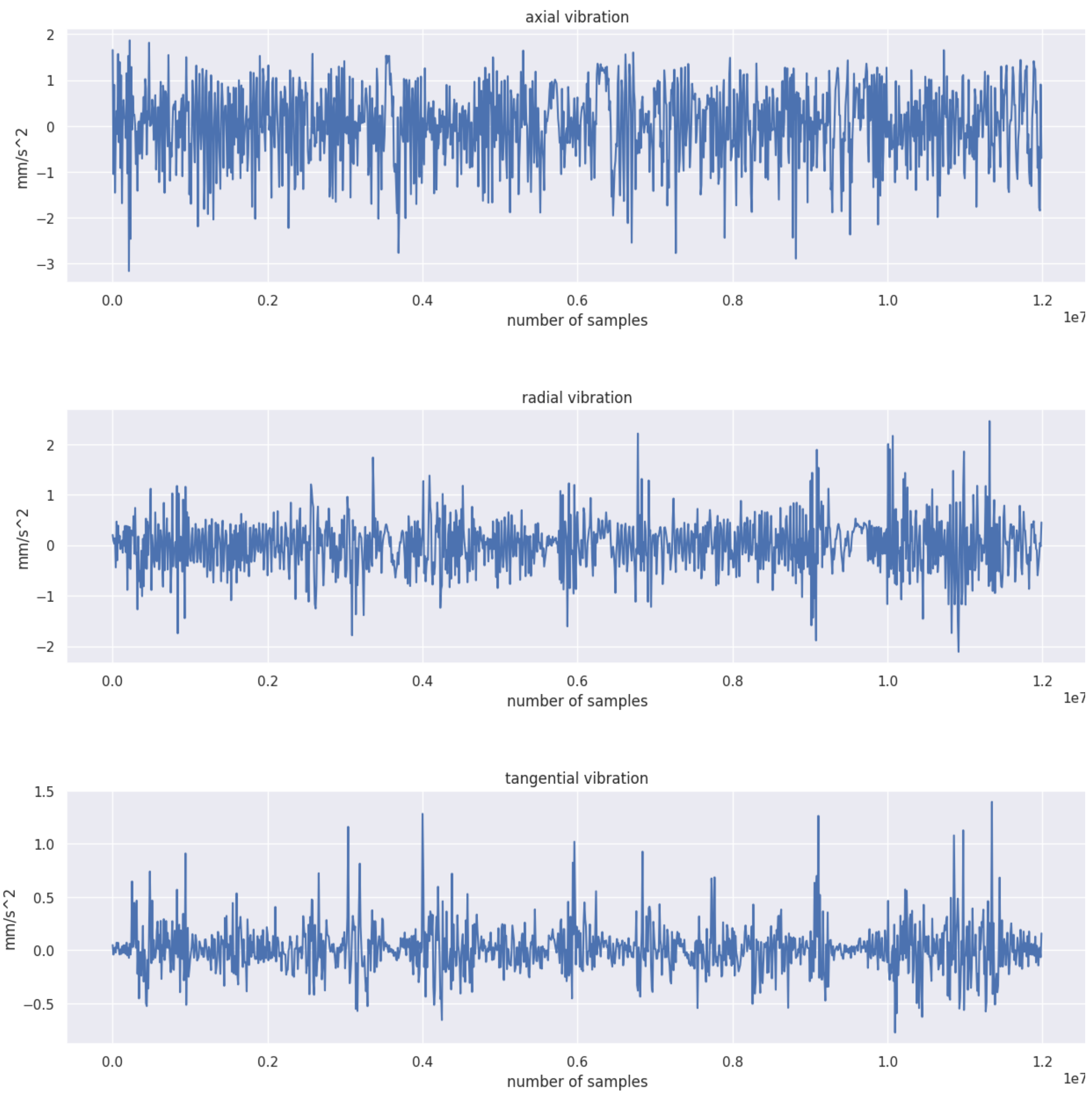


Figure III.3 Vibrations of 6g imbalance state

We notice that the 6g vibrations are slightly more important than the normal state vibrations, where there is no weight added to the system, which is logical considering that the amount of weight added here is not significant.

We continue to plot the 20g imbalance state:

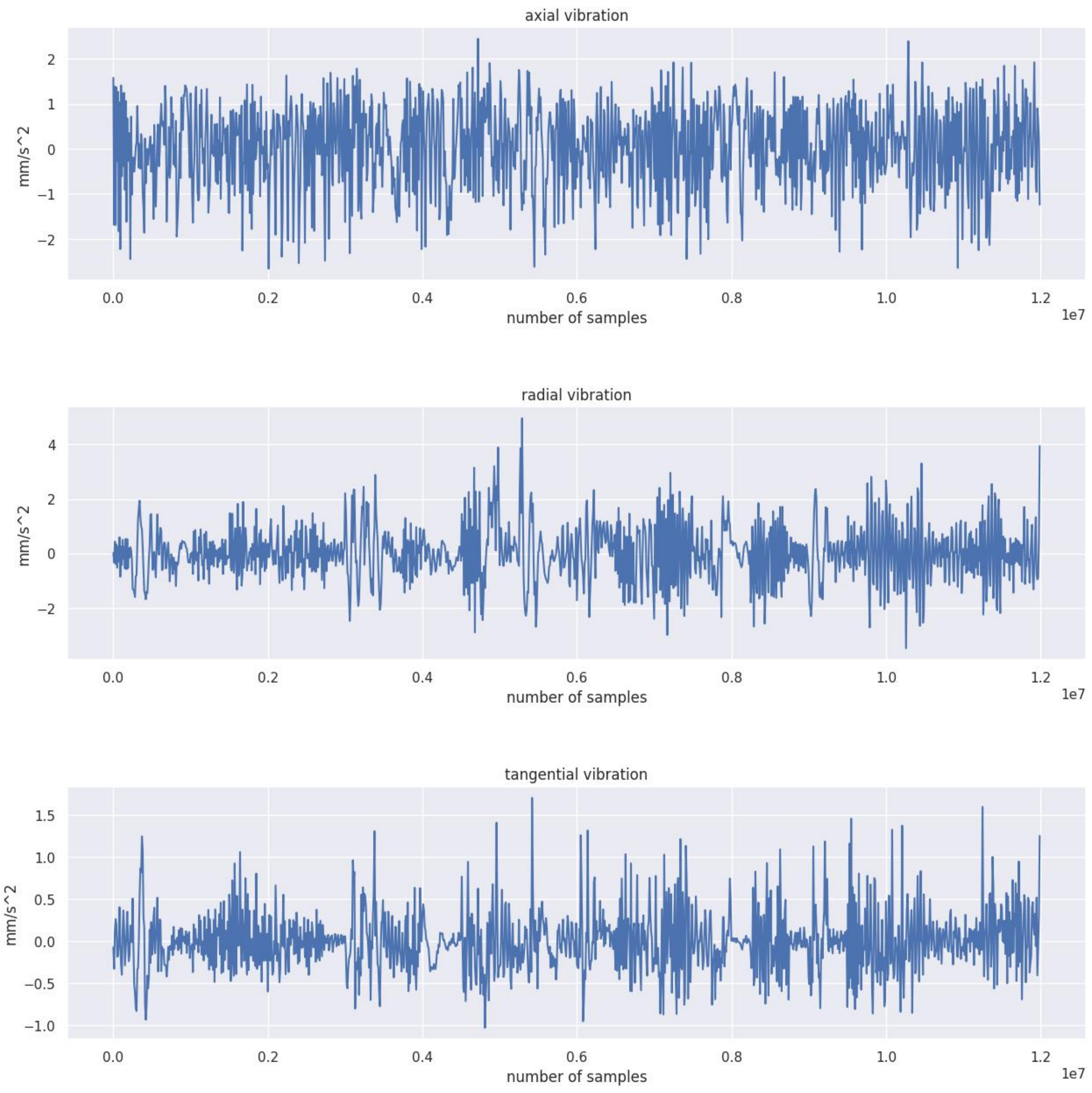


Figure III.4 Vibrations of 20g imbalance state

We can see that the vibrations after adding 20g of weight are considerable compared to the normal state and the 6g imbalance states' vibrations. These results suggest that the more weight we add to the system, the more vibrations we are going to have, which is logically acceptable.

We proceed to our last plot of vibrations with the 35g imbalance state:

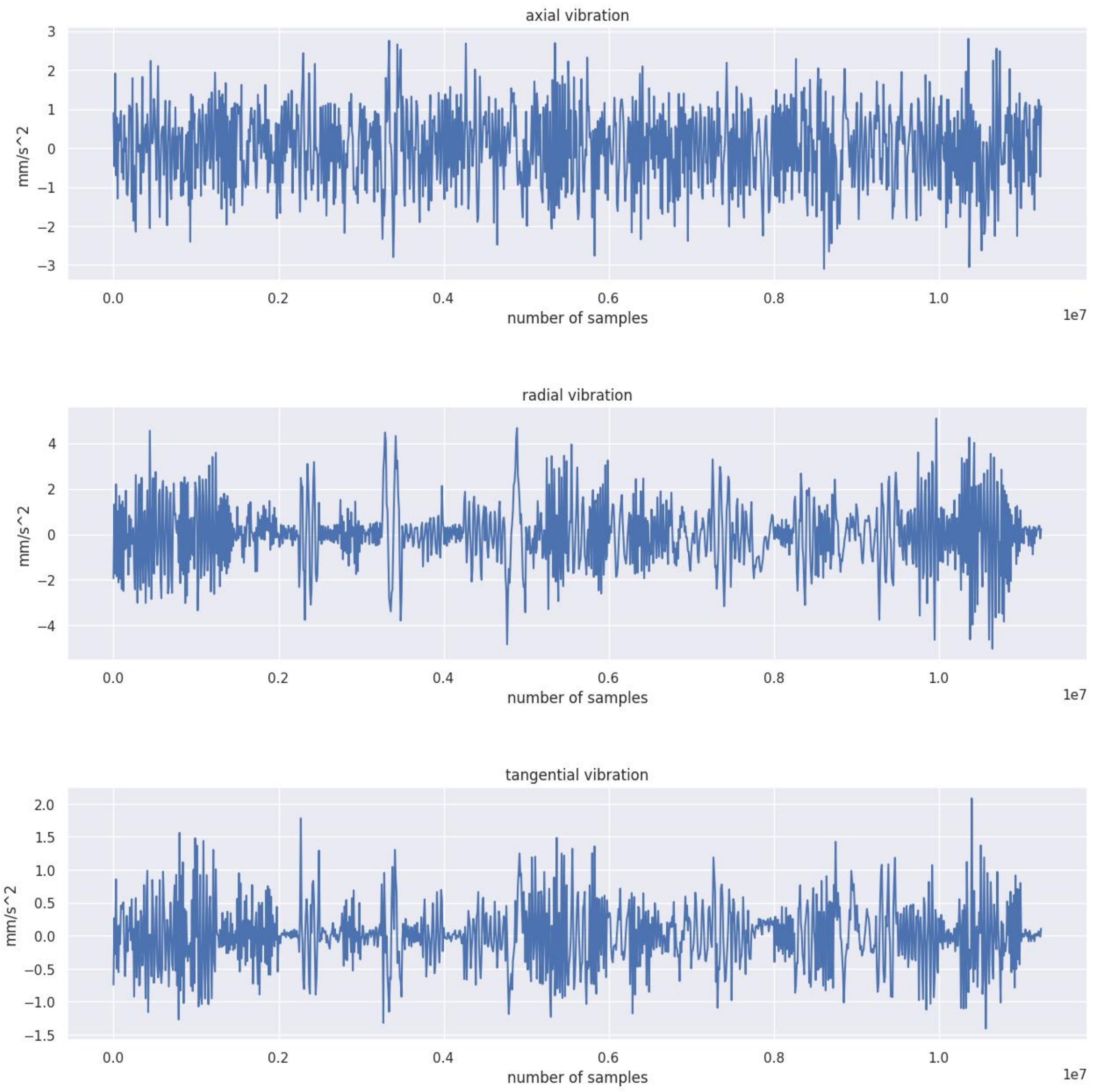


Figure III.5 Vibrations of 35g imbalance state

We can notice here that the 35g imbalance state have significantly more vibrations than the other states, which is due to the added weight difference.

After plotting and visualizing all the vibrations from different states, we conclude that the amount of weight added can directly affect the importance of the systems' vibrations, and we confirm that the more weight we add to our system, the more vibrations are going to occur.

III.3.2.3 Data split

To effectively train our model, we need to split the data into a training set and a test set.

- **The training set** is the portion of the dataset on which the model learns the underlying patterns and relationships between the input features and the target variable. The training set is typically larger than the test set to provide enough data for the model to learn from.
- **The test set** is a subset of the data that is used to evaluate the performance of the trained model. It serves as an unseen dataset that the model has not been exposed to during the training phase. The test set is used to assess how well the model generalizes to new, unseen data. By making predictions on the test set, the model's performance metrics, such as accuracy, precision, recall, or mean squared error, can be evaluated. The test set helps determine the effectiveness and reliability of the trained model.

With the following line of code, we perform the train-test split on our dataset using the “train_test_split” function from the scikit-learn library:

```
data = data_n.copy()
from sklearn.model_selection import train_test_split

X_train, X_test = train_test_split(data, test_size=0.05,
random_state=42)
train = X_train
test = X_test
print("Shape of Train Data : {}".format(train.shape))
print("Shape of Test Data : {}".format(test.shape))

# Shape of Train Data : (11637500, 3)
# Shape of Test Data : (612500, 3)
```

The shape of the training set and split set vary according to the amount of data used. In general, the larger the data the more percentage we can use for the training set, in our case, we used 12 million data values, which allowed us to do a 95% to 5% train-test split.

III.3.3 Data preprocessing

After downloading, visualizing, and finally splitting the data, it is time to start the preprocessing step. Preprocessing is a necessary step applied to transform the raw data into a format that maximizes the performance and reliability of our model.

III.3.3.1 Down-sampling

Down-sampling is a preprocessing technique often used when the dataset is too large to be handled effectively. It aims to reduce the amount of data treated in order to create a more balanced dataset, which can improve the performance and fairness of the DL model.

With the following lines of code, we define the “downSampler” function, an implementation of a down-sampling technique that reduces the size of a given dataset by calculating the mean of consecutive subsets of the data:

```
def downSampler(data, a, b):
    '''
    data is the dataset we want to sample
    a is the start index
    b is the sampling rate
    '''
    data_decreased = pd.DataFrame()
    x = b
    for i in range(int(len(data)/x)):
        data_mean = data.iloc[a:b,:].sum() / x
        data_decreased = pd.concat([data_decreased,
data_mean.to_frame().T])
        a += x
        b += x
    return data_decreased

train = downSampler(train,0,1000)
test = downSampler(test,0,1000)
print('train data after down sampling: ',train.shape)
print('test data after down sampling: ', test.shape)

# train data after down sampling:  (11637, 3)
# test data after down sampling: (612, 3)
```

Using the “downSampling” function, we down-sample the train and test data with a sampling rate of 1000, the code reduces the size of both datasets by aggregating consecutive subsets of 1000 samples into single rows.

III.3.3.2 Reshaping

LSTM models are primarily designed to work with three-dimensional data format, thus, to benefit from the full potential of our model, we will reshape the 2D data we have into a 3D data using the following code lines:

```
X_train = train.values.reshape(train.shape[0], 1, train.shape[1])
X_test = test.values.reshape(test.shape[0], 1, test.shape[1])

print('Training data shape:', X_train.shape)
print('Test data shape:', X_test.shape)

# Training data shape: (11637, 1, 3)
# Test data shape: (612, 1, 3)
```

After executing, the “X_train” and “X_test” variable will contain our final datasets, down-sampled, reshaped, and ready to implement in our model.

III.3.4 Autoencoder model

After preparing and preprocessing the data, it is time to introduce our Autoencoder model. We will explain how we created the model, trained it, and finally visualize it.

III.3.4.1 Creating the model

We create our model by defining the function “standard_autoencoder” with the following lines of code:

```
def standard_autoencoder(X):

    input_data = Input(shape=(X.shape[1], X.shape[2]))
    encoded = Dense(units=128, activation='relu', 
kernel_regularizer=regularizers.l1(0.01))(input_data)
    encoded = Dense(units=128, activation='relu', 
kernel_regularizer=regularizers.l2(0.01))(encoded)
    encoded = Dropout(0.3)(encoded)
    decoded = Dense(X_train.shape[2], activation='sigmoid')(encoded)
    autoencoder = Model(input_data, decoded)
    return autoencoder
```

The **standard_autoencoder** function defines a standard autoencoder model architecture that takes a three-dimensional tensor as input. The model architecture consists of several layers:

- Input Layer: Takes the input data tensor.
- Encoding Layers: Two fully connected (dense) layers with 128 units each, using ReLU activation function. These layers aim to learn a compressed representation (encoding) of the input data.
- Dropout Layer: A dropout layer is applied to the encoded representation with a dropout rate of 0.3. Dropout helps to prevent overfitting by randomly dropping out units during training, forcing the model to learn more robust and generalized representations.

- Decoding Layer: A dense layer with the same number of units as the input data's feature dimension and using the sigmoid activation function. It aims to reconstruct the original input data from the encoded representation.
- Autoencoder Model: The model is defined by specifying the input and output layers.

After defining a function that creates our model, we will now configure its training process with the following code:

```
autoencoder = standard_autoencoder(X_train)
autoencoder.compile(optimizer='adam', loss='mse')
early_stopping = EarlyStopping(patience=20, monitor='val_loss',
restore_best_weights=True)
autoencoder.summary()
```

Our autoencoder model is created using the **standard_autoencoder** function with **X_train** as the input. The autoencoder is compiled with the Adam optimizer which is an efficient optimization algorithm for neural networks, and the mean squared error (MSE) is used as the loss function to measure the discrepancy between the model's predictions and the true values., as it computes the average squared difference between the predicted and target values. The accuracy metric is also specified to evaluate the model's performance.

The MSE function is defined as follows:

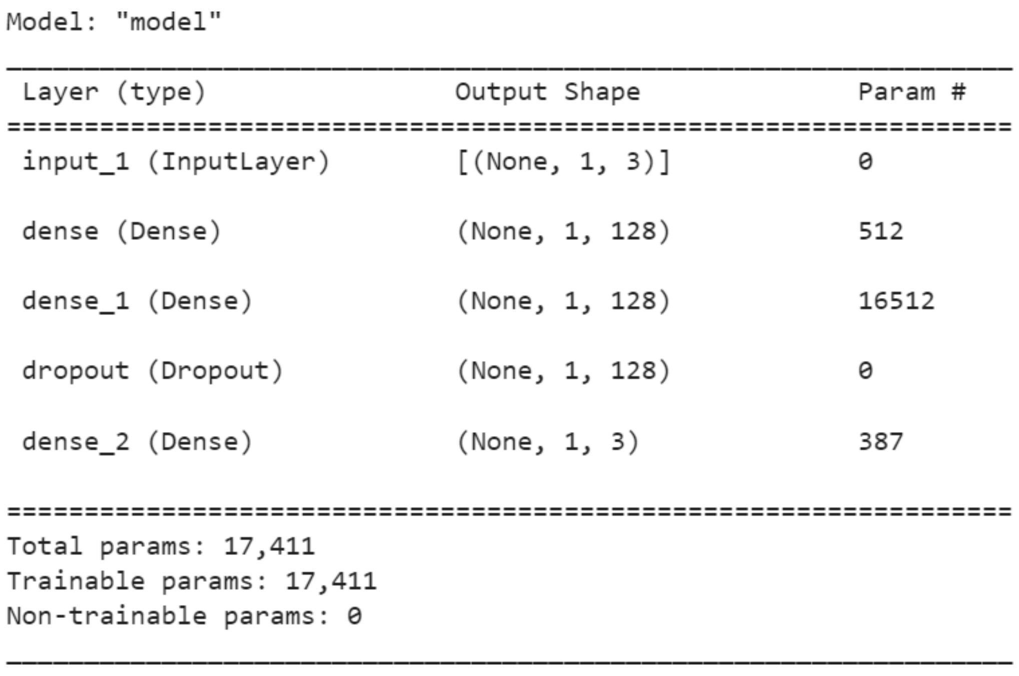
$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \qquad \text{III.1}$$

Where:

- **n** is the number of samples
- **y** is the target value
- **ŷ** is the predicted value

To monitor the training progress and prevent overfitting, an early stopping callback is defined. This callback tracks the validation loss (**val_loss**) and halts the training process if no improvement is observed for a specified number of epochs (20 in our case) based on the **patience** parameter.

The **summary()** method is then called on the autoencoder model to display a summary of its architecture, including the number of parameters and the shape of each layer. The summary provides useful information about the model's structure, allowing us to verify the number of trainable parameters and check for any potential issues or discrepancies.



```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 1, 3)]            0

 dense (Dense)               (None, 1, 128)            512

 dense_1 (Dense)             (None, 1, 128)            16512

 dropout (Dropout)           (None, 1, 128)            0

 dense_2 (Dense)             (None, 1, 3)              387

=================================================================
Total params: 17,411
Trainable params: 17,411
Non-trainable params: 0
_________________________________________________________________
```

Figure III.6 Summary of the AE model

The summary of the autoencoder model shows the architecture and the number of parameters. The model consists of four layers:

- InputLayer: The input layer receives data with a shape of (None, 1, 3), indicating a batch size of None (variable), a single time step, and three features.
- Dense: This layer is a fully connected layer with 128 units. It takes the input from the previous layer and produces an output of shape (None, 1, 128). The number of parameters in this layer is 512.
- Dense_1: Another fully connected layer with 128 units follows. It takes the previous layer's output as input and produces an output of shape (None, 1, 128). The number of parameters in this layer is 16,512.
- Dropout: A dropout layer is applied with a rate of 0.3, which randomly sets 30% of the input units to 0 at each training step. It helps in preventing overfitting.
- Dense_2: The final dense layer has 3 units, corresponding to the original shape of the data. It produces an output of shape (None, 1, 3). The number of parameters in this layer is 387.

The total number of trainable parameters in the model is 17,411.

III.3.4.2 Training the model

After visualizing our model and before starting the training process, we define a custom callback class called **TrainingTimeCallback** that tracks the training time for each epoch during the model training. The callback records the time taken for each epoch and stores it in the **training_times** list attribute.

```
class TrainingTimeCallback(Callback):
    def __init__(self):
        self.training_times = []

    def on_epoch_begin(self, epoch, logs=None):
        self.start_time = time.time()

    def on_epoch_end(self, epoch, logs=None):
        end_time = time.time()
        epoch_time = end_time - self.start_time
        self.training_times.append(epoch_time)

# Create an instance of the TrainingTimeCallback
training_time_callback = TrainingTimeCallback()
```

By using this custom callback during model training, we can access the **training_times** list to analyze and evaluate the training time for each epoch, which can be useful for performance monitoring and optimization purposes.

After that, we finally start the training process of our model using the **fit** method, as shown in the following lines:

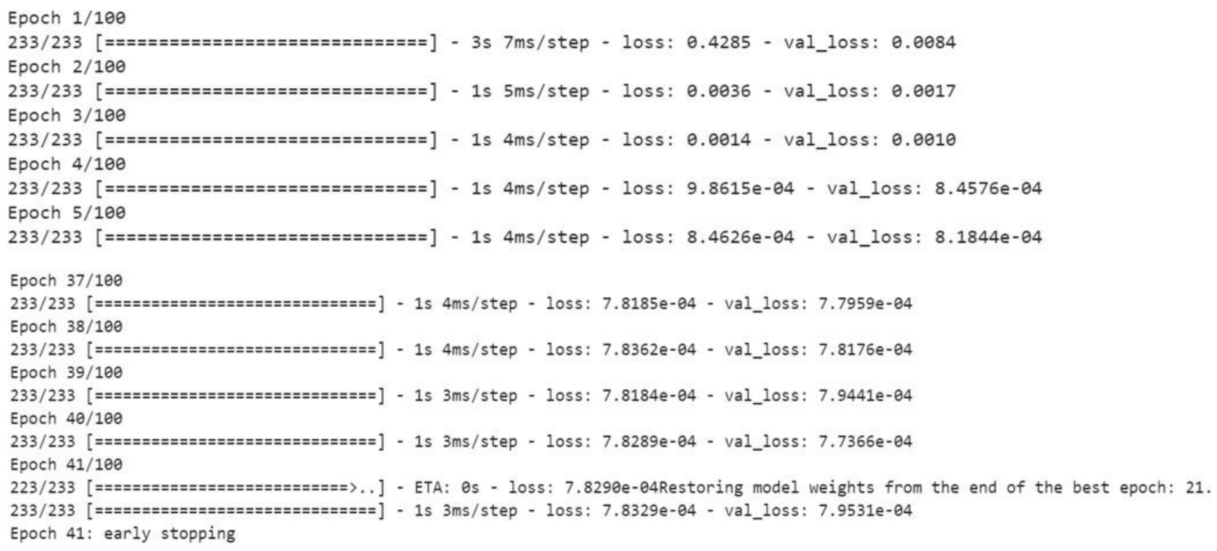
```
nb_epochs = 100
batch_size = 40

history = model.fit(X_train, X_train, epochs=nb_epochs,
batch_size=batch_size, validation_split=0.2, verbose=1,
callbacks=[early_stopping,training_time_callback])
```

The training process is executed with the following parameters:

- X_train: The input training data, which consists of the original sequences that the model will learn to reconstruct.
- X_train: The target training data, which is set to be the same as the input data. The goal of the Autoencoder is to reconstruct the input sequences.
- Epochs = nb_epochs: The number of training epochs, specifying how many times the model will iterate over the entire training dataset.

- batch_size = batch_size: The number of samples per gradient update. The training data is divided into smaller batches, and the model's weights are updated after processing each batch.
- validation_split = 0.2: The fraction of the training data to be used as validation data. In this case, 20% of the training data will be used for validation during training.
- Verbose = 1: Controls the verbosity of the training process. Setting it to 1 displays a progress bar and training information during each epoch.
- Callbacks = [early_stopping, training_time_callback]: Specifies the list of callbacks to be used during training. In this case, the early_stopping callback and the custom training_time_callback are included.

To summarize, the **model.fit** method starts the training process and returns a history object, which contains information about the training metrics and loss values for each epoch. This object can be used to analyze and visualize the training progress and performance of the model.



```
Epoch 1/100
233/233 [==============================] - 3s 7ms/step - loss: 0.4285 - val_loss: 0.0084
Epoch 2/100
233/233 [==============================] - 1s 5ms/step - loss: 0.0036 - val_loss: 0.0017
Epoch 3/100
233/233 [==============================] - 1s 4ms/step - loss: 0.0014 - val_loss: 0.0010
Epoch 4/100
233/233 [==============================] - 1s 4ms/step - loss: 9.8615e-04 - val_loss: 8.4576e-04
Epoch 5/100
233/233 [==============================] - 1s 4ms/step - loss: 8.4626e-04 - val_loss: 8.1844e-04

Epoch 37/100
233/233 [==============================] - 1s 4ms/step - loss: 7.8185e-04 - val_loss: 7.7959e-04
Epoch 38/100
233/233 [==============================] - 1s 4ms/step - loss: 7.8362e-04 - val_loss: 7.8176e-04
Epoch 39/100
233/233 [==============================] - 1s 3ms/step - loss: 7.8184e-04 - val_loss: 7.9441e-04
Epoch 40/100
233/233 [==============================] - 1s 3ms/step - loss: 7.8289e-04 - val_loss: 7.7366e-04
Epoch 41/100
223/233 [===========================>..] - ETA: 0s - loss: 7.8290e-04Restoring model weights from the end of the best epoch: 21.
233/233 [==============================] - 1s 3ms/step - loss: 7.8329e-04 - val_loss: 7.9531e-04
Epoch 41: early stopping
```

Figure III.7 First & last five AE Epochs

Figure III.7 shows us the first and last five epochs of the training process, we can see that the training time of the last epochs is shorter than the training time of the last ones, the loss values are also decreasing which indicates the model is learning and performing well. The training stopped at epoch 41 due to the early-stopping parameter used, showing that the best epoch performance was the 21st epoch.

III.3.4.3 Visualizing the model

We continue by plotting the epochs' training time, which will allow us to identify any significant variations or trends in training time and provide insights into the efficiency of the training process.

We can plot the training time by executing the following code:

```
plt.plot(training_time_callback.training_times)
plt.title('Training Time per Epoch')
plt.xlabel('Epoch')
plt.ylabel('Time (seconds)')
plt.show()

# calculate the total training time
training_times = training_time_callback.training_times
total_time = np.sum(training_times)
print('Total training time:', total_time, 'seconds')
```

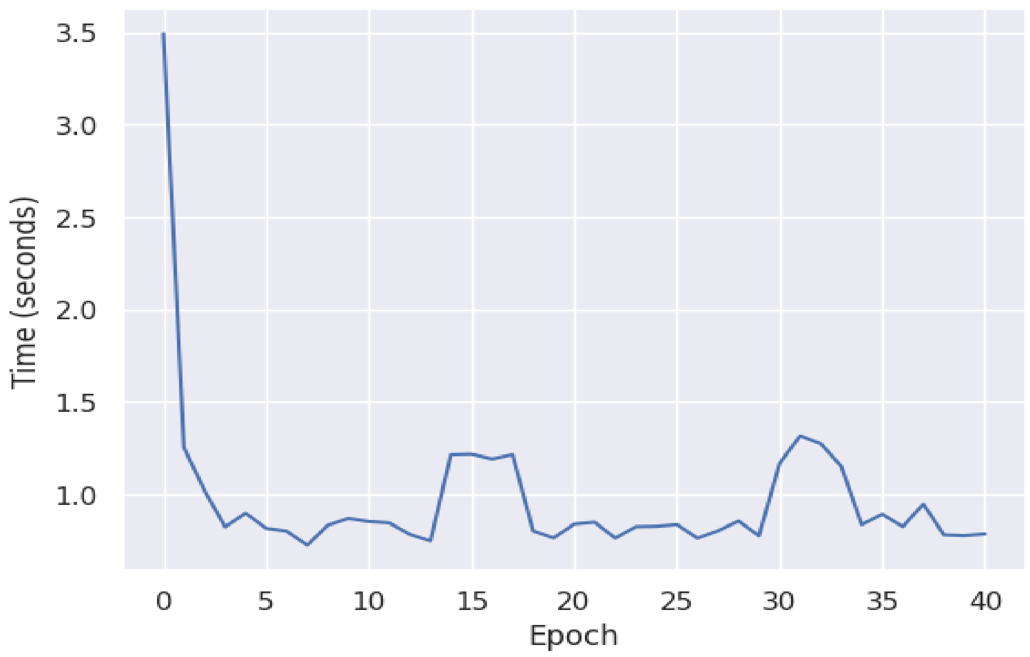


Figure III.8 AE Training Time per Epoch

The total training time was only 40.07 seconds due to the early-stopping parameter that prevented the model from continuing unnecessary training by stopping the training process at epoch 41. We can see that the training time decreases significantly during the first epochs, and fluctuates from around the 15th epoch until the last one. This means that our model reached its optimal capacities in the first epochs, and continuing the training will not result in major improvements.

To further clarify our results, we will plot the model loss with the following code:

```
fig, ax = plt.subplots(figsize=(14, 6), dpi=80)
ax.plot(history.history['loss'], 'b', label='Train', linewidth=2)
ax.plot(history.history['val_loss'], 'r', label='Validation',
linewidth=2)
ax.set_title('Model Loss', fontsize=18)
ax.set_ylabel('Loss (MSE)')
ax.set_xlabel('Epoch')
ax.legend(loc='upper right')
plt.show()
```

The plot allows us to visualize the model's performance in terms of how well it is reducing the loss function during training and validation. We can see that the training loss decrease and reach stability within the first epochs, overlapping with the validation loss. It means that the model has learned the underlying patterns and is able to make accurate predictions on both the training and validation datasets. It quickly adapts to the data and reduces the loss, reaching its full potential early on, suggesting that continuing the training process may only result in minor improvements.

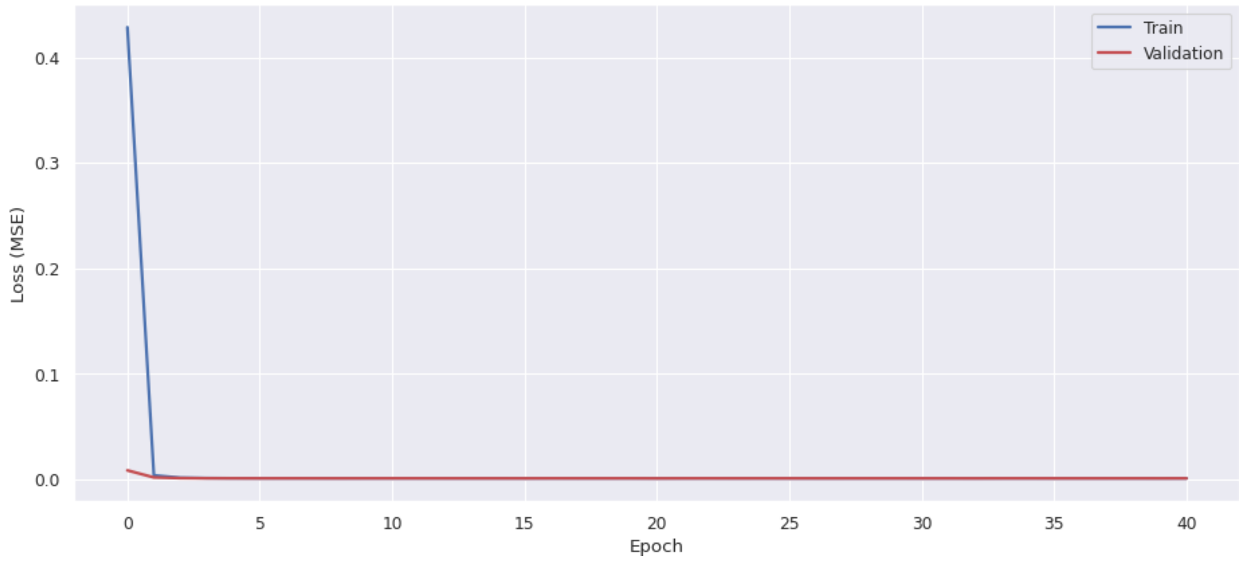


Figure III.9 Autoencoder Model Loss

We conclude by visualizing the MSE for each vibration (axial, radial, and tangential) in the predicted output compared to the original test data. Using the following code:

```
fig, axs = plt.subplots(num_features, 1, figsize=(15, 10),
sharex=False)
```

```
for i in range(num_features):
    feature_mse
    axs[i].plot(feature_mse)
    axs[i].set_ylabel('MSE')
    axs[i].set_xlabel('Data Point Index')
    axs[i].set_title('MSE for {}'.format(feature_names[i]))


plt.subplots_adjust(hspace=0.5)
plt.show()
```

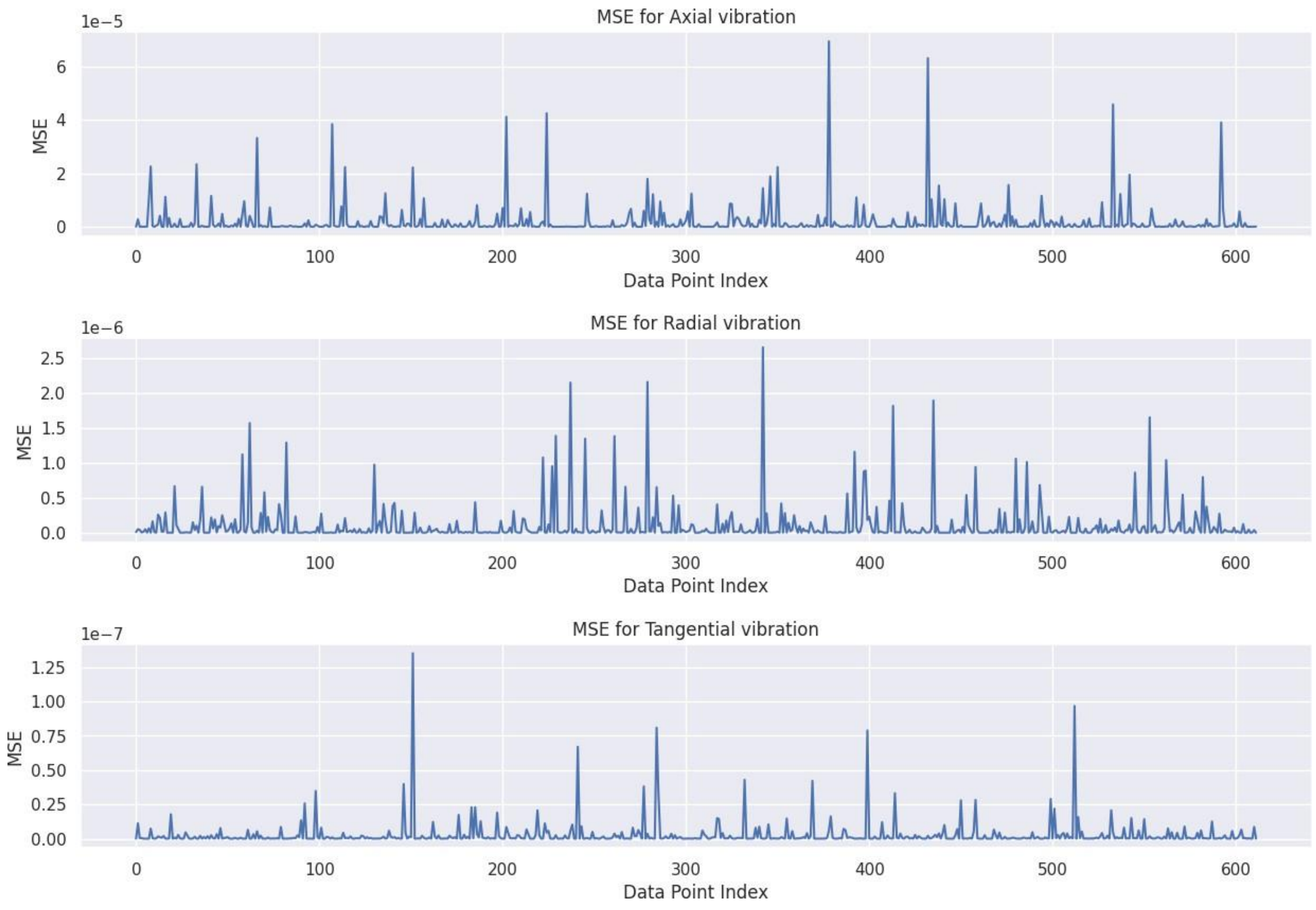


Figure III.10 Autoencoder MSE

By plotting the MSE, we can now assess how well the autoencoder model is reconstructing each individual feature. We notice that the MSE values are very small (from 10^{-5} to 10^{-7}), which indicates that the predicted values are closer to the original values, implying better reconstruction accuracy and overall performance.

The sudden peaks are anomalies that can be treated using a threshold, which is the subject of the next title.

III.3.5 AE Anomaly Detection

After preparing, explaining, and visualizing our AE model, we finally reach the anomaly detection step.

We begin by plotting anomalies of the 6g imbalance data with the following code:

```
fig, axs = plt.subplots(num_features, 1, figsize=(15, 10),
sharex=False)
feature_names = ['Axial vibration', 'Radial vibration', 'Tangential
vibration']

for i in range(num_features):
    feature_mse_6g = np.square(data_6g[:, 0, i] - predict_data_6g[:,0,
i])
    threshold_6g = np.percentile(feature_mse_6g, 95)
    axs[i].axhline(threshold_6g, color='red', linestyle='--',
label='Threshold')
    axs[i].plot(feature_mse_6g)
    axs[i].set_ylabel('MSE')
    axs[i].set_xlabel('Data Point Index')
    axs[i].set_title('MSE for {}'.format(feature_names[i]))
    axs[i].legend(loc='upper right')
    print('Threshold for {} is:'.format(feature_names[i]),threshold_6g)

plt.subplots_adjust(hspace=0.5)
plt.show()
```

This code allows us to plot the MSE for each vibration of the 6g imbalance data, overlaying a threshold line on the plot. The purpose of this is to visually identify data points that have MSE values above the threshold, which indicates anomalies or deviations from the expected behavior.

The threshold is calculated using the 95th percentile of the feature MSE distribution. This means that 95% of the MSE values fall below this threshold, and the remaining 5% are considered potential anomalies.

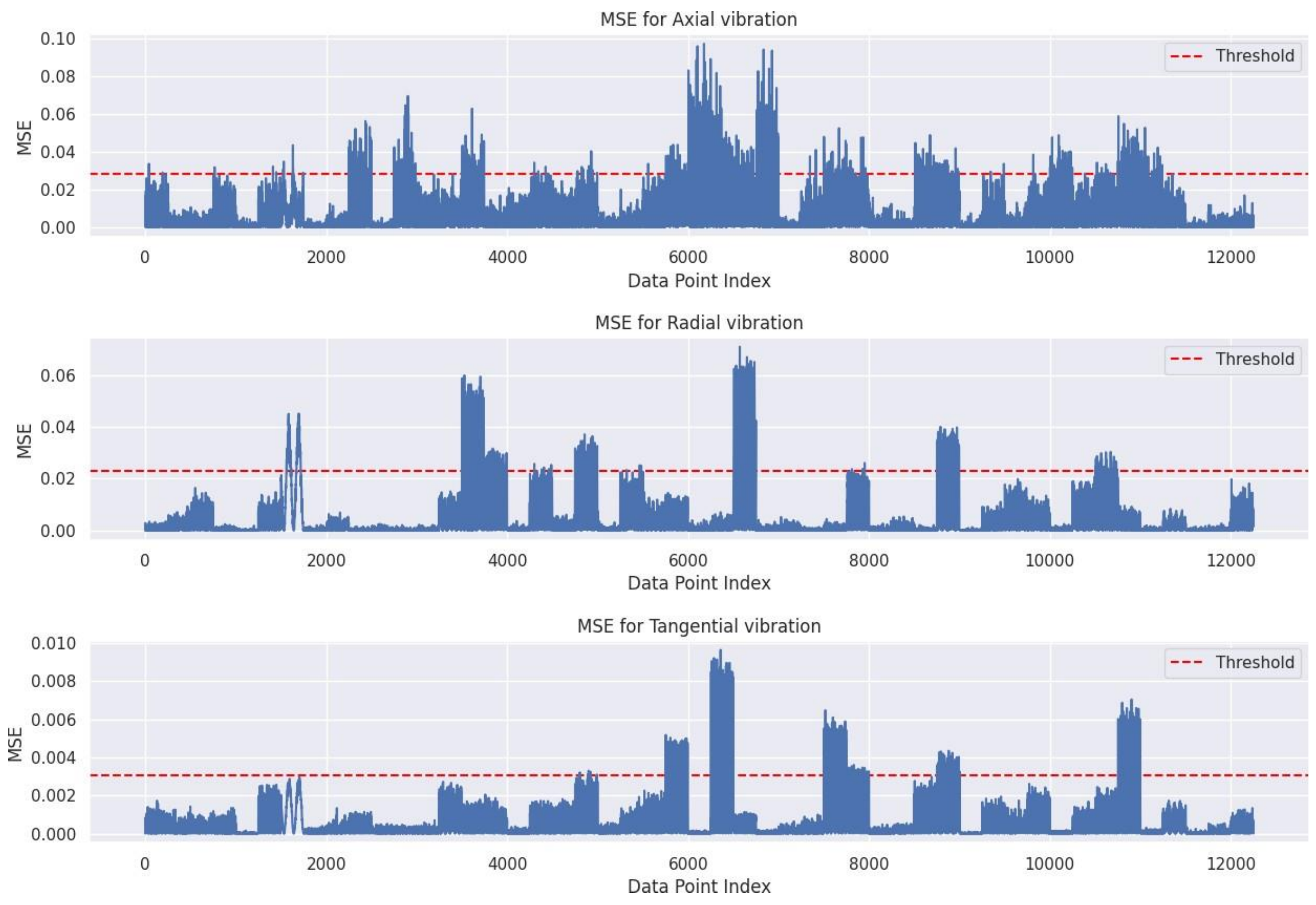


Figure III.11 AE 6g Imbalance Anomalies

Using the 95th percentile as a threshold offers a balanced approach. Higher percentiles create a more conservative threshold, reducing false positives but potentially missing some anomalies. Lower percentiles increase sensitivity to anomalies but may result in more false positives. The selection of the threshold depends on the specific application and the desired level of sensitivity and precision.

By setting the threshold at the 95th percentile, we can capture a majority of the normal data while allowing a small portion of anomalies.

Using the same code and threshold, we plot the MSE for the 20g and 35g imbalances.

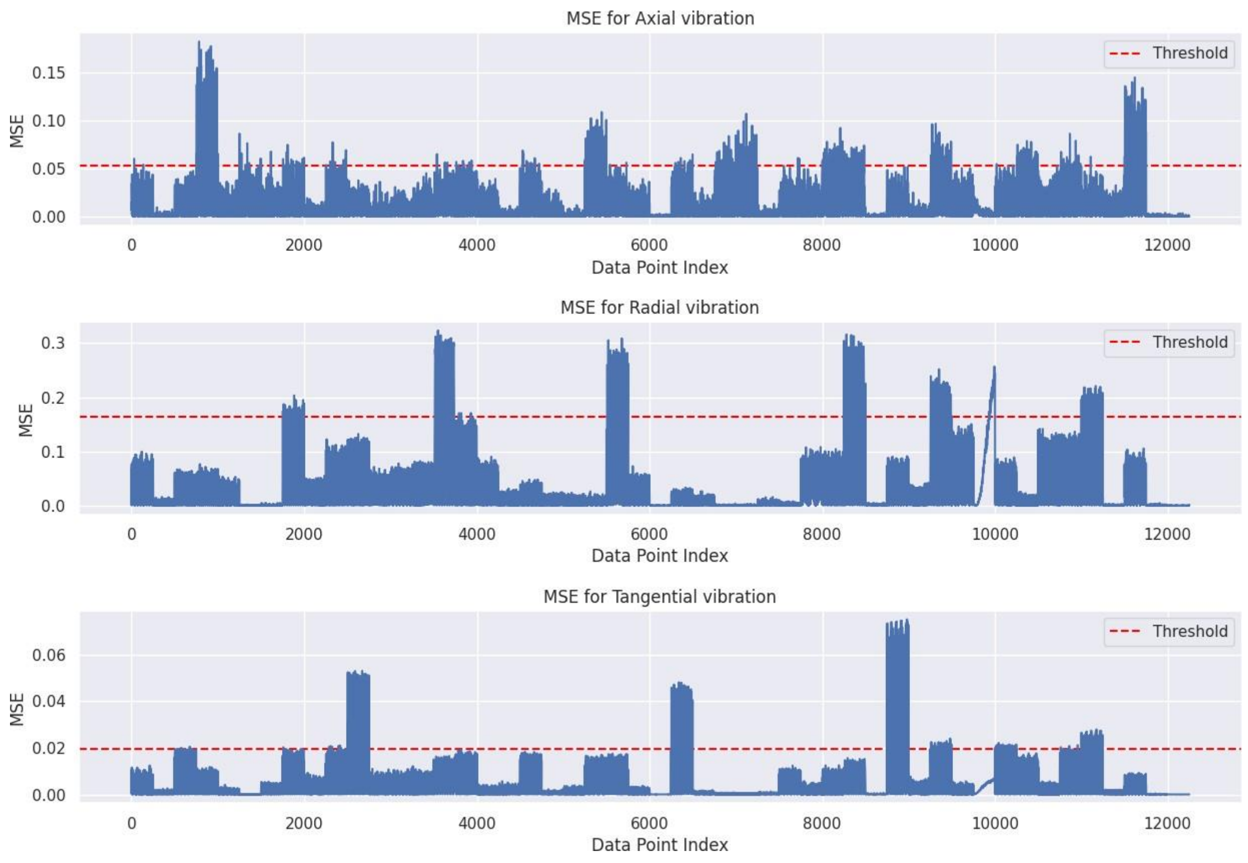


Figure III.12 AE 20g Imbalance Anomalies

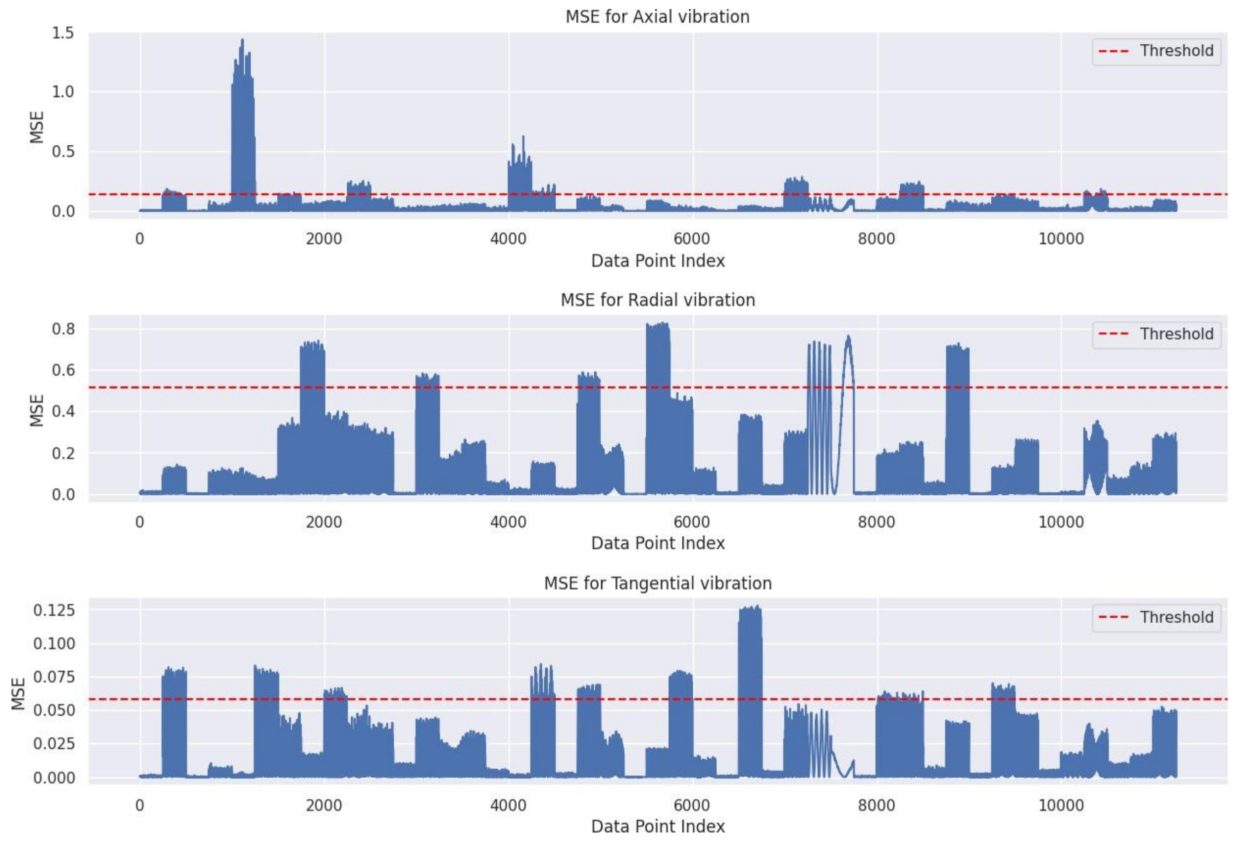


Figure III.13 AE 35g Imbalance Anomalies

We can notice from the MSE imbalance plots that the 35g anomalies are more important than the 20g anomalies, and the 20g anomalies are more important than the 6g ones, which means that increasing the imbalances weight led to increased anomalies, due to the increased vibrations.

In other words, increasing imbalance weights in the rotating components of the machinery will increase vibrations, and these vibrations lead to more errors, and thus, more anomalies are detected.

III.4 Conclusion

In this chapter, we introduced and explained our database, our Autoencoder model, and the program used to create, and visualize the model. We discussed and clarified the results by several different plots and figures. Lastly, we integrated the anomaly detection approach using a defined threshold and applied it to the different imbalances from our database.

We conclude that the imbalance weights have a direct exponential relationship with the amount of vibrations and potential anomalies.

Chapter IV

LSTM-Autoencoder model

Chapter IV
LSTM-Autoencoder model

IV.1 Introduction

After introducing our Autoencoder model and explaining in detail every part of it, in this chapter, we review the LSTM Autoencoder-based model and compare it with the regular Autoencoder. The performance and accuracy of both models trained on the same data will be compared and the obtained results will be clarified.

IV.2 LSTM Autoencoder-based program

Similar to our regular Autoencoder-based program proposed in Chapter Three, we used the same lines of code to download, prepare and preprocess the same database used on the first model. Thus, we will directly introduce the LSTM Autoencoder model.

IV.2.1 Creating the model

We create our model using the function “LSTM_AE” which is defined as the following:

```
def LSTM_AE_model(X):
  inputs = Input(shape=(X.shape[1], X.shape[2]))
  L1 = LSTM(64, activation='relu', return_sequences=True,
            kernel_regularizer=regularizers.l2(0.00))(inputs)
  L2 = LSTM(64, activation='relu',return_sequences=False)(L1)
  L3 = RepeatVector(X.shape[1])(L2)
  L4 = LSTM(32, activation='relu',return_sequences=True)(L3)
  L5 = LSTM(64, activation='relu',return_sequences=True)(L4)
  output = TimeDistributed(Dense(X.shape[2]))(L5)
  model = Model(inputs = inputs, outputs = output)
  return model
```

In this architecture, the input data **X** is passed through several LSTM layers. The first LSTM layer (**L1**) processes the input data, returning sequences to preserve the temporal information. The second LSTM layer (**L2**) further processes the output from the first layer, but does not return sequences. Instead, it compresses the information into a fixed-length vector. Line **L3** creates a layer that repeats the compressed representation of the input sequence, allowing subsequent LSTM layers to process it and generate a reconstructed sequence of the same length as the original input. The third LSTM layer (**L4**) takes this compressed representation and reconstructs a sequence of the same length as the original input.

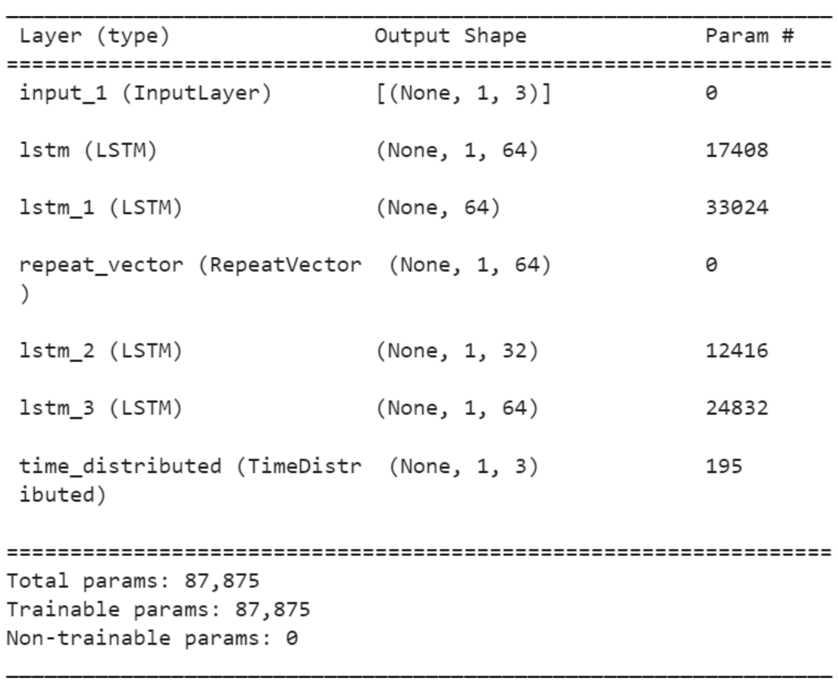
Finally, the fourth LSTM layer (**L5**) refines the reconstructed sequence. The output layer applies a dense transformation to each time step independently using the **TimeDistributed** wrapper, aiming to reconstruct the original input data. The resulting model is an LSTM Autoencoder that learns to compress and reconstruct the input data while capturing temporal dependencies and patterns.

After defining a function that creates our model, we configure the training process of the LSTM Autoencoder model, including the optimizer, loss function, metrics, and early stopping callback, and provide a summary of the model's architecture with the following code:

```
model = LSTM_AE_model(X_train)
model.compile(optimizer='adam', loss='mse', metrics=['accuracy'])
# Define early stopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=20,
verbose=1, mode='min', restore_best_weights=True)
model.summary()
```

The provided lines of code perform the necessary setup and configuration for training an LSTM Autoencoder model, using the Adam optimizer and MSE loss function the same way as for the Autoencoder model.

The early stopping is defined with a number of 20 epochs for the **patience**, and the model summary is finally displayed.



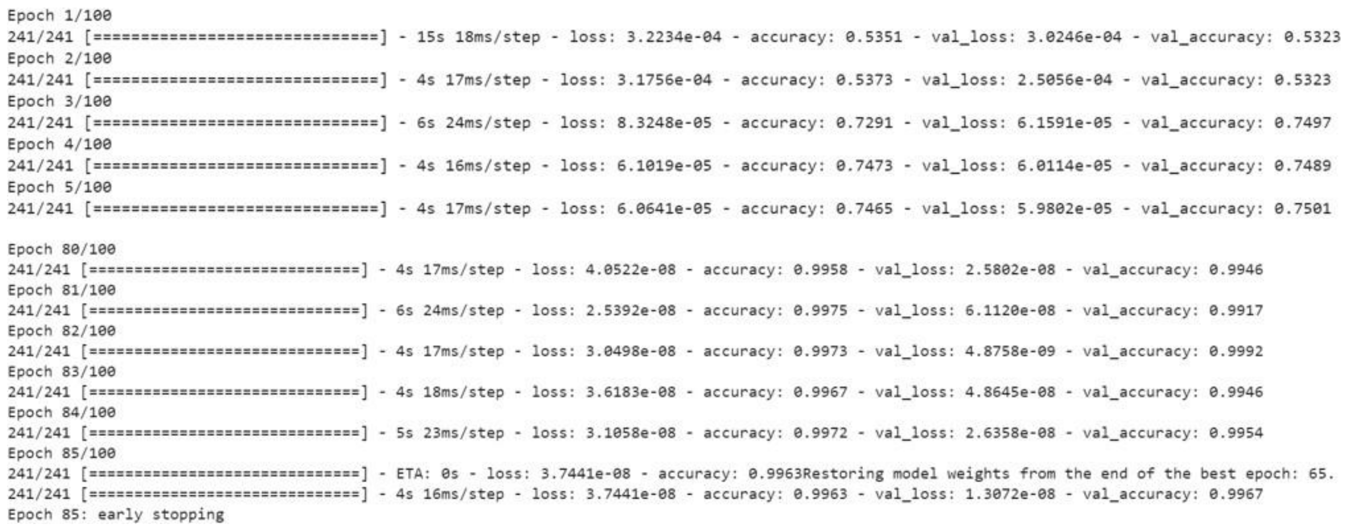
```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 1, 3)]            0

 lstm (LSTM)                 (None, 1, 64)             17408

 lstm_1 (LSTM)               (None, 64)                33024

 repeat_vector (RepeatVector  (None, 1, 64)            0
 )

 lstm_2 (LSTM)               (None, 1, 32)             12416

 lstm_3 (LSTM)               (None, 1, 64)             24832

 time_distributed (TimeDistr  (None, 1, 3)             195
 ibuted)

=================================================================
Total params: 87,875
Trainable params: 87,875
Non-trainable params: 0
_________________________________________________________________
```

Figure IV.1 Summary of LSTM Autoencoder model

IV.2.2 Training the model

After displaying the model, we proceed to use the custom callback used in the previous chapter to access the training time of each epoch, and then we start the training.

The **model.fit** method starts the training process and returns a history object, which contains information about the training metrics and loss values for each epoch.



```
Epoch 1/100
241/241 [==============================] - 15s 18ms/step - loss: 3.2234e-04 - accuracy: 0.5351 - val_loss: 3.0246e-04 - val_accuracy: 0.5323
Epoch 2/100
241/241 [==============================] - 4s 17ms/step - loss: 3.1756e-04 - accuracy: 0.5373 - val_loss: 2.5056e-04 - val_accuracy: 0.5323
Epoch 3/100
241/241 [==============================] - 6s 24ms/step - loss: 8.3248e-05 - accuracy: 0.7291 - val_loss: 6.1591e-05 - val_accuracy: 0.7497
Epoch 4/100
241/241 [==============================] - 4s 16ms/step - loss: 6.1019e-05 - accuracy: 0.7473 - val_loss: 6.0114e-05 - val_accuracy: 0.7489
Epoch 5/100
241/241 [==============================] - 4s 17ms/step - loss: 6.0641e-05 - accuracy: 0.7465 - val_loss: 5.9802e-05 - val_accuracy: 0.7501

Epoch 80/100
241/241 [==============================] - 4s 17ms/step - loss: 4.0522e-08 - accuracy: 0.9958 - val_loss: 2.5802e-08 - val_accuracy: 0.9946
Epoch 81/100
241/241 [==============================] - 6s 24ms/step - loss: 2.5392e-08 - accuracy: 0.9975 - val_loss: 6.1120e-08 - val_accuracy: 0.9917
Epoch 82/100
241/241 [==============================] - 4s 17ms/step - loss: 3.0498e-08 - accuracy: 0.9973 - val_loss: 4.8758e-09 - val_accuracy: 0.9992
Epoch 83/100
241/241 [==============================] - 4s 18ms/step - loss: 3.6183e-08 - accuracy: 0.9967 - val_loss: 4.8645e-08 - val_accuracy: 0.9946
Epoch 84/100
241/241 [==============================] - 5s 23ms/step - loss: 3.1058e-08 - accuracy: 0.9972 - val_loss: 2.6358e-08 - val_accuracy: 0.9954
Epoch 85/100
241/241 [==============================] - ETA: 0s - loss: 3.7441e-08 - accuracy: 0.9963Restoring model weights from the end of the best epoch: 65.
241/241 [==============================] - 4s 16ms/step - loss: 3.7441e-08 - accuracy: 0.9963 - val_loss: 1.3072e-08 - val_accuracy: 0.9967
Epoch 85: early stopping
```

Figure IV.2 First & last five LSTM-AE Epochs

Figure IV.2 shows us that the last epochs have a better loss and accuracy compared to the first ones, which means that our model performance has improved during training. By using the early-stop method, our model stopped the training at epoch 85, mentioning and keeping the best epoch values (epoch 65), and thus, the training process is concluded.

IV.2.3 Visualizing the model

To better clarify our results, we will plot and further explain them, starting from the training time, model loss, model accuracy, and finally the models' MSE.

IV.2.3.1 Training Time

We plot the epochs' training time to see how our model performs. The total training time was 355.83 seconds, which is approximately 6 minutes.

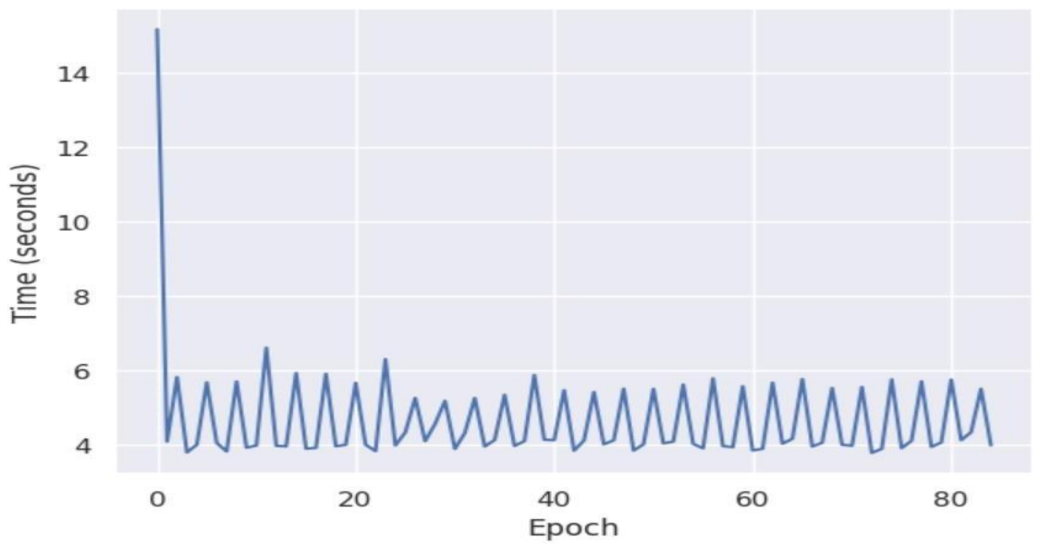


Figure IV.3 LSTM-AE Training Time per Epoch

By analyzing the plot result, we notice that the model quickly converges to a relatively optimal solution. This indicates that the model has learned the underlying patterns and features of the training data efficiently within the initial epochs.

The stagnation of training time from around the 40th epoch suggests that further training does not significantly enhance the model's performance, which is why our model saved the 65th epoch as the best one and stopped the training at the 80th epoch.

IV.2.3.2 Model Loss

The model loss represents the difference between the reconstructed sequences (output) generated by the LSTM Autoencoder model and the original input sequences (target).

The loss value is calculated using the MSE as the loss function, the plot of the model loss over the training epochs provides insights into how the loss changes as the model undergoes training.

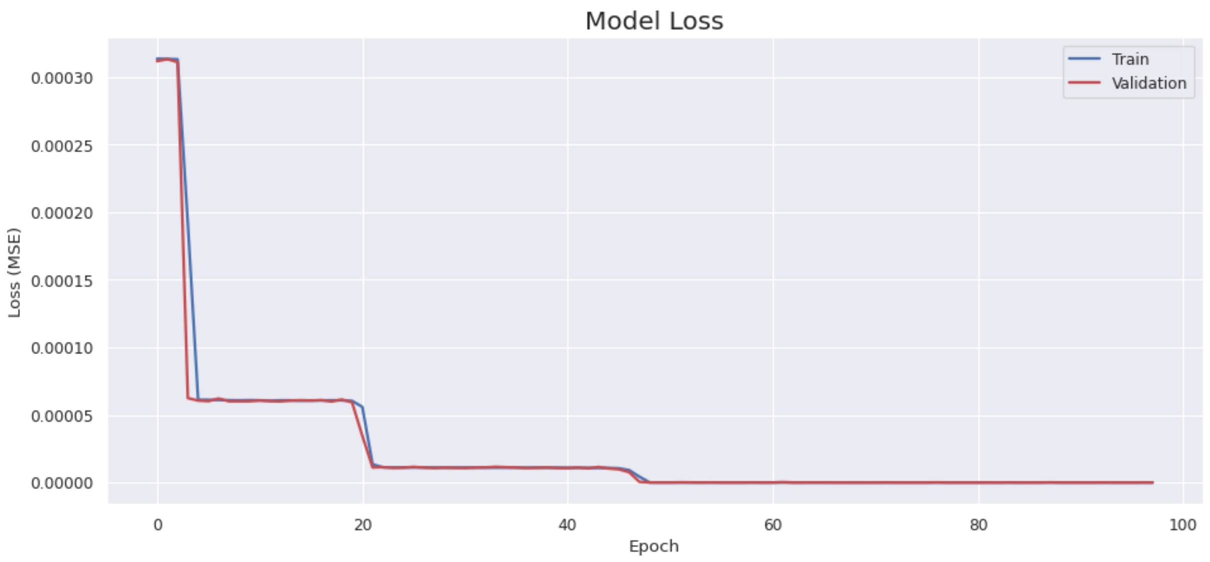


Figure IV.4 LSTM-AE Model Loss

By monitoring the loss, we can evaluate the model's learning progress and convergence. A decreasing loss indicates that the model is learning to reconstruct the sequences effectively, which is the case of our model.

IV.2.3.3 Model Accuracy

We pursue our plotting phase with the next code that will access the accuracy values and plot the model accuracy:

```
# Access the accuracy values from the training history

accuracy =history.history['accuracy']

val_accuracy = history.history['val_accuracy']

# Plot the accuracy
plt.plot(accuracy, label='Training Accuracy')
plt.plot(val_accuracy, label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Model Accuracy')
plt.legend()
plt.show()
```

The plot of training accuracy and validation accuracy provides valuable insights into the model's performance during training. By comparing the two curves, we can assess the model's ability to learn and generalize.

In our plot, we can see both lines increasing and converging, indicating that the model is learning well and generalizing to unseen data. A large gap between the two lines may suggest overfitting, which is not our case. We can also observe the overall trend of the curves, with increasing accuracy over time indicating successful learning.

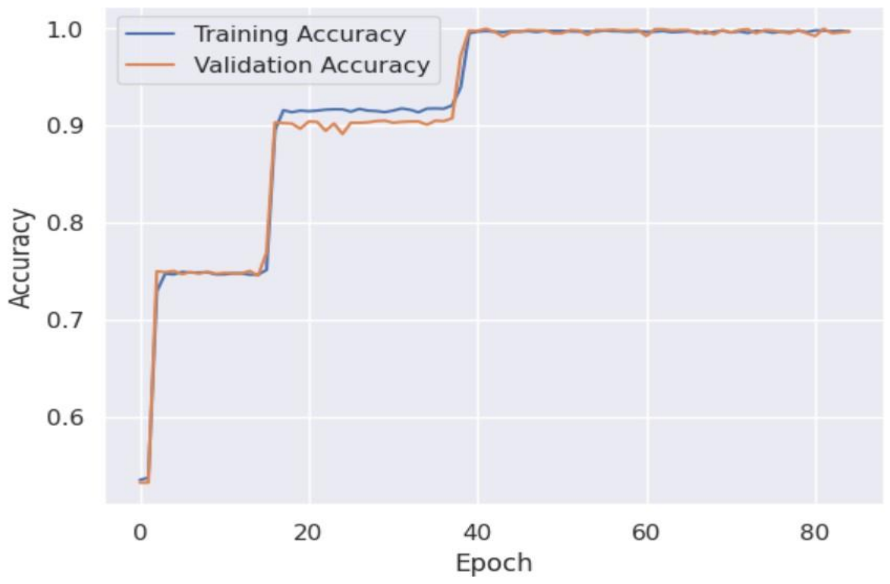


Figure IV.5 LSTM-AE Model Accuracy

By comparing the accuracy and the loss plots, we can observe that the model's accuracy increases while the loss decreases. This indicates that the model is effectively optimizing its predictions and learning from the training data.

IV.2.3.4 Model Mean Squared Error

We conclude the LSTM-AE visualization by plotting the models' mean squared error of the imbalance axial, radial, and tangential vibrations using the same lines of code introduced in Chapter 3.

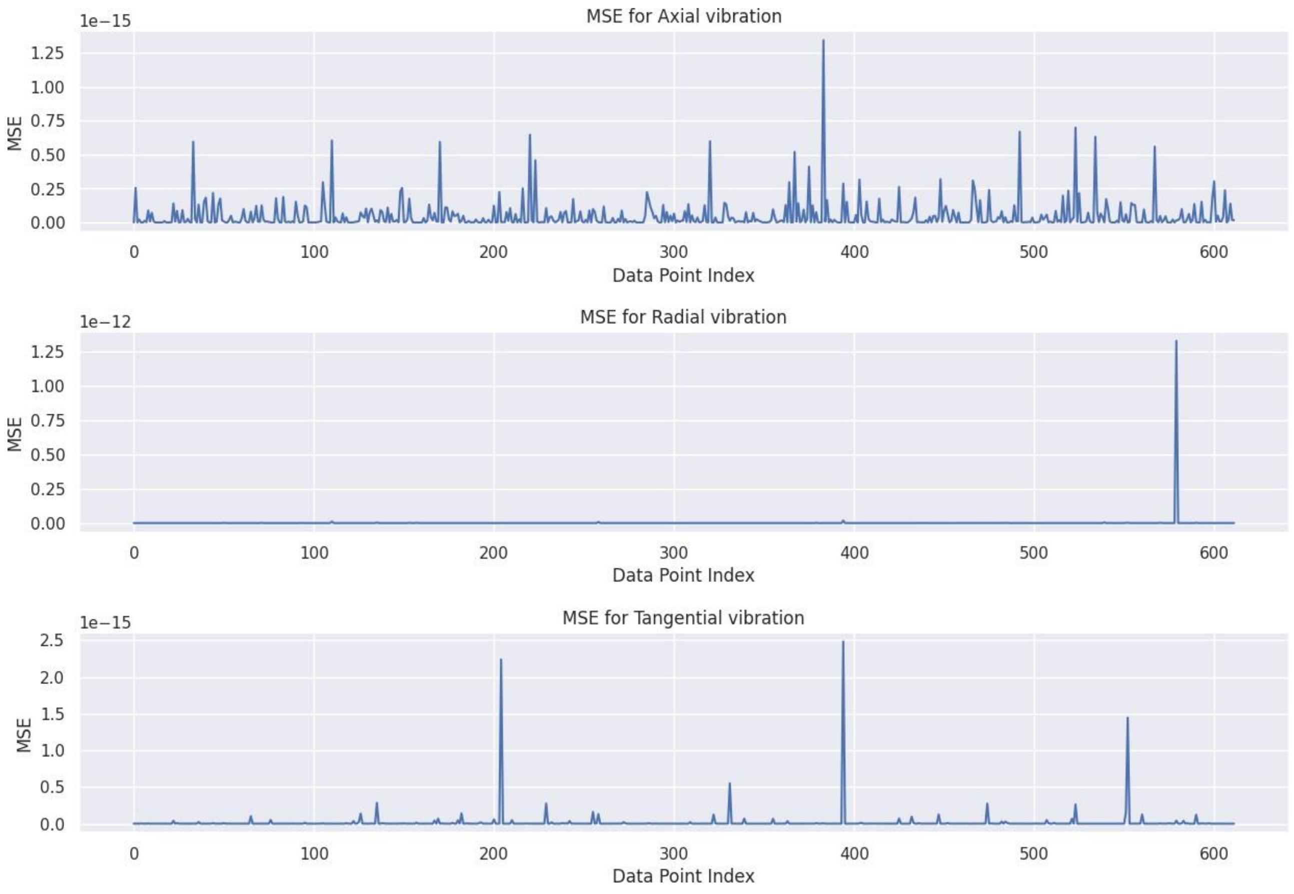


Figure IV.6 LSTM-AE MSE

We can see that the MSEs of the model are very small, which means that our model learned effectively and its performance is acceptable. The random peaks of MSE indicate the presence of anomalies which will be detected next.

IV.2.4 LSTM-AE Anomaly detection

After explaining and visualizing the LSTM-AE model, we will now review its performance in detecting anomalies.

We will plot the 6g, 20g, and 35g MSE imbalances and apply a 95% threshold the same way we did with the regular AE model in the previous chapter.

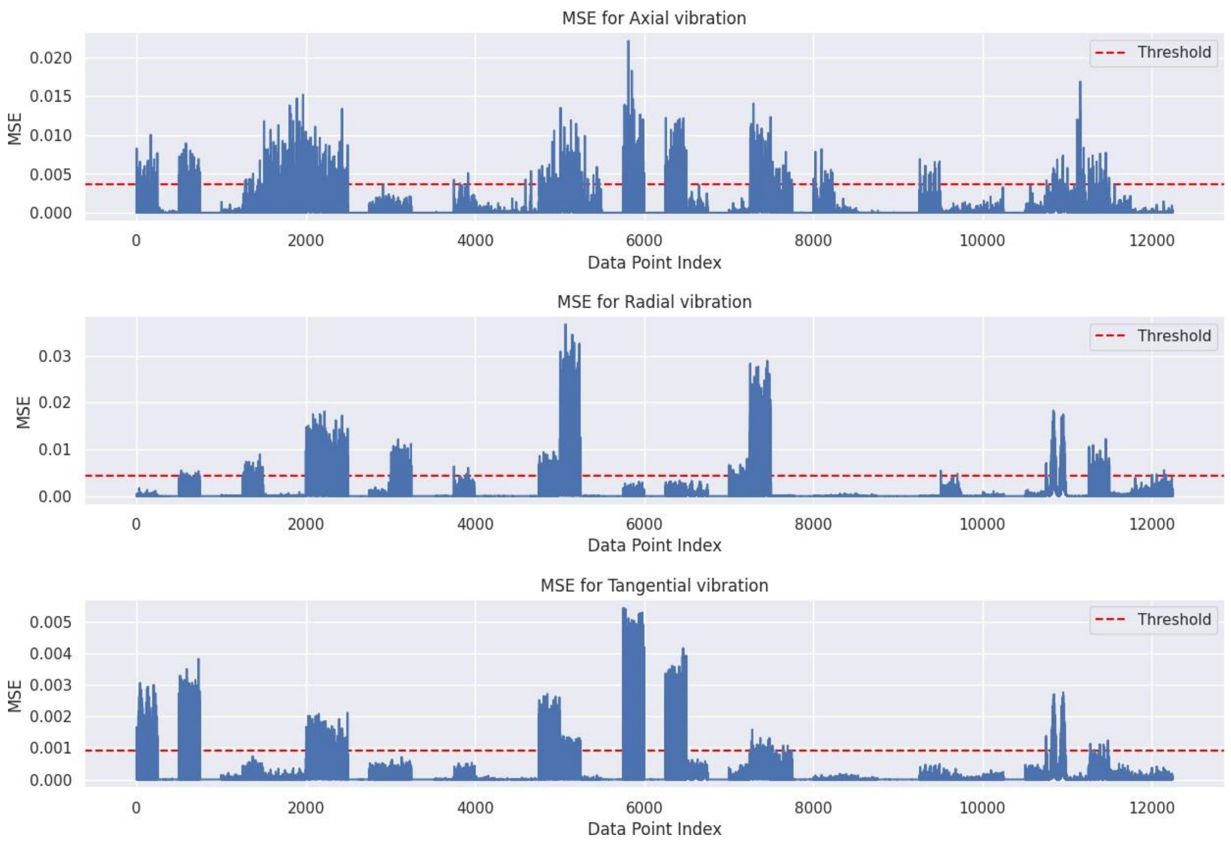


Figure IV.7 LSTM-AE 6g Imbalance Anomalies

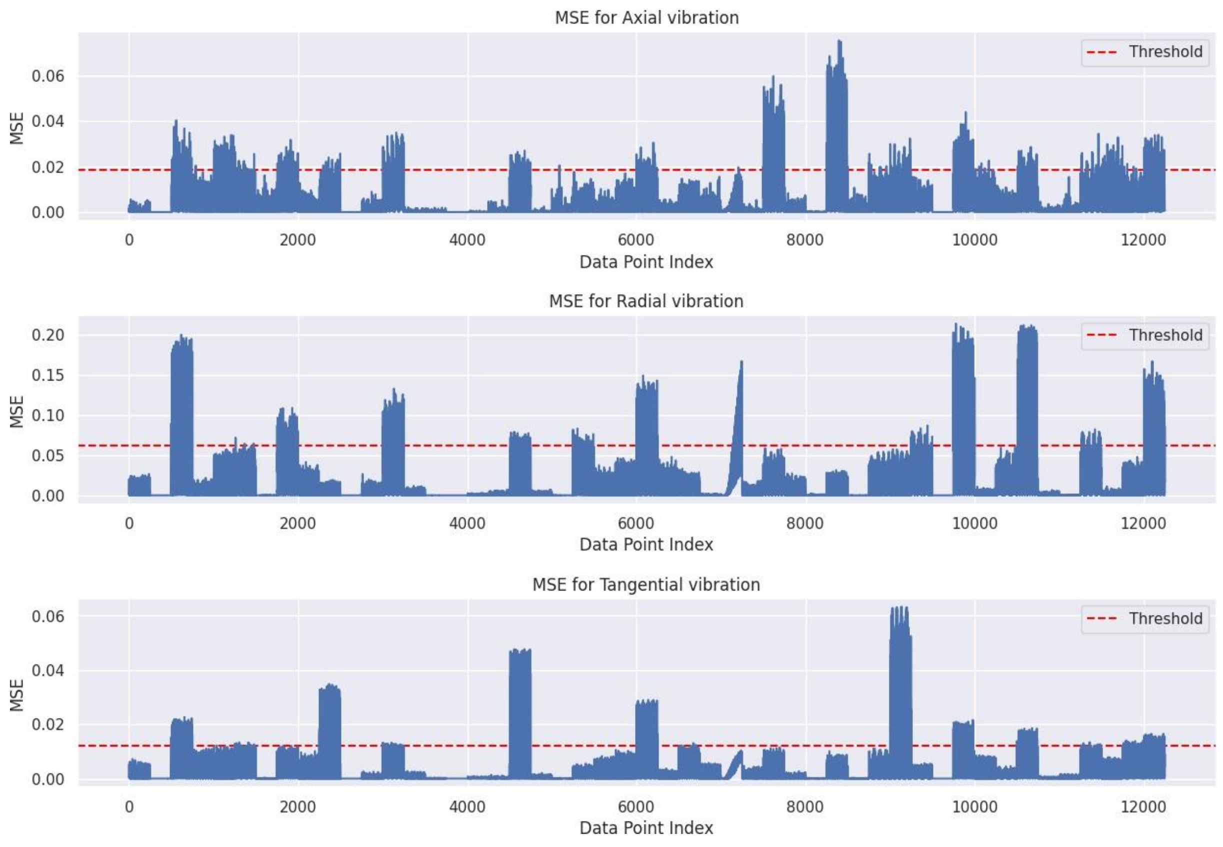


Figure IV.8 LSTM-AE 20g Imbalance Anomalies

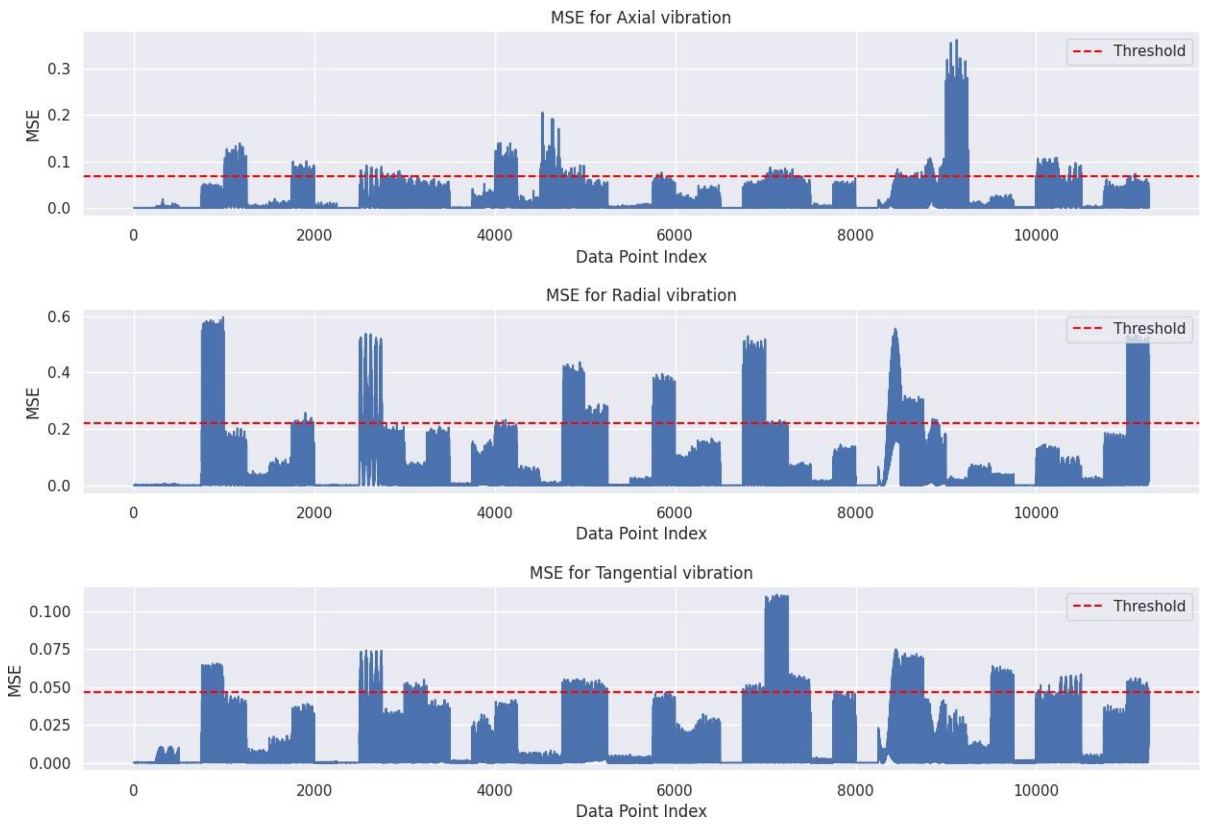


Figure IV.9 LSTM-AE 35g Imbalance Anomalies

We can see that the LSTM-AE imbalances act the same way as the regular AE imbalances, meaning that the heaviest weight vibrations are the most important ones perceived, which confirms the conclusion stated in chapter 3 about the exponential relationship between the weights added to the rotation components of the machinery, the vibrations, and the anomalies resulted.

IV.3 LSTM-AE vs Regular AE

After introducing both models, we will now compare their performance by reviewing three aspects:

- Training time
- Loss functions
- MSE Anomalies

IV.3.1 Training Time

After plotting the training time of both models on the same data, we notice that the training process of the LSTM-AE took a significant amount of time compared to the training process of the regular AE (6 minutes vs 40 seconds). This can be explained by the more complex architecture of the LSTM-AE.

The LSTM-AE requires more time for each epoch due to the additional computations involved in training the LSTM layers. These computations include the forward and backward propagation of information through the recurrent connections and updating of the LSTM cell states.

Consequently, the overall training process takes longer compared to the regular AE, which has a simpler architecture and fewer computational operations.

IV.3.2 Loss functions

The MSE loss functions of both models decrease significantly with time and reach a plateau suggesting that both models successfully learned the data patterns and reached their optimal performance.

On the other hand, the MSE loss values of the LSTM-AE were remarkably less important than the loss values of the regular AE (0.0003 vs 0.4), which proves the superiority of the LSTM-AE in handling large complex amount of data and detect temporal features and dependencies.

The LSTM layers allow the model to learn and exploit the temporal relationships between the input features. This enables the LSTM-AE to better reconstruct the input data and minimize the reconstruction error, as quantified by the MSE loss function.

In contrast, the regular AE lacks the ability to explicitly model and capture temporal dependencies. It treats the input data as independent and identically distributed samples, neglecting any underlying sequential information. As a result, the regular AE may struggle to effectively reconstruct the time-dependent patterns in the data, leading to higher MSE loss values.

By leveraging the memory cells and recurrent connections, the LSTM-AE is able to better preserve the temporal information and reconstruct the input data with higher fidelity, resulting in lower MSE loss values. This highlights the advantage of using LSTM-based architectures when dealing with sequential or time-dependent data.

IV.3.3 MSE Anomalies

While both models had impressive low mean squared error values on the different axes of the weighted imbalances, the MSEs of the LSTM-AE were significantly smaller than the ones on the regular AE (10^{-15} vs 10^{-7}).

This means that while both models are performing well, the LSTM-AE performances are superior due to its complexity and capability to handle big amounts of data and temporal dependencies, which is confirmed by the loss results discussed previously.

The remarkably smaller values of the MSE in the LSTM-AE also indicates that the anomalies present and detected in the machinery are less important and less common compared to the anomalies in the regular AE model, which only solidate and confirms our results and findings.

IV.4 Conclusion

After introducing our LSTM-AE model, explain it, and finally visualizing it, we compared its performance to the regular AE model proposed in Chapter 3.

We found that due to the complexity of its architecture and additional computational operations present in the LSTM layers of the LSTM-AE, the regular AE offers faster training time, while the LSTM-AE provides superior performance in terms of MSE and its ability to capture intricate temporal dependencies.

In the end, the choice between the two models depends on the specific requirements of the application, weighing the trade-off between training time and performance.

General conclusion

In this work, we talked about industrial maintenance in a general manner and showcased how and why Predictive Maintenance is generally superior when it comes to industrial maintenance approaches due to its ability to predict failures, minimize downtime, and improve the reliability of the machinery.

Artificial Intelligence plays a significant role in the field of PdM, AI techniques and methodologies are employed to analyze large volumes of data, extract meaningful insights, and make accurate predictions about the health and performance of industrial equipment. Therefore, we introduced Artificial Intelligence and all its subfields, including Machine Learning, Artificial Neural Networks, Deep Learning, and finally Autoencoders and Long Short-Term Memory architectures used in our model.

Our model presented a combination of the two architectures, LSTM layers were added to the Autoencoder in order to leverage the LSTM capacity for handling large amounts of temporal data.

To prove the efficiency of the model, we first introduced a regular Autoencoder and trained both models on the same data using the same code written with Python. After visualizing the results and competence of the two models, we compared and reviewed their performance on three points: Training time, Loss function, and MSE anomalies. We found that the LSTM-Autoencoder had significantly smaller loss values (0.0003 vs 0.4) and MSE anomalies (10^{-15} vs 10^{-7}) compared to the regular Autoencoder, while the regular Autoencoder outperformed the LSTM when it comes to training time (40 seconds vs 6 minutes). We concluded that the LSTM-Autoencoder had superior performance although it was slower than the regular Autoencoder due to the complexity of the LSTM layers added.

In spite of the proven efficiency of the LSTM-Autoencoder model in predictive maintenance and anomaly detection, there are several deep learning techniques developed and constantly being improved at this time, due to the increased popularity of AI in general and the continuous implementation of this field in the industrial sector in particular.

Finally, the most appropriate DL model or approach may vary depending on the systems' characteristics, specific requirements, data features, and goals of the PdM application. That being the case, it is recommended to experiment and compare different models to determine the most effective approach for a particular use case.

Bibliography

Bibliography

[1] Çınar, Z. M., Abdussalam Nuhu, A., Zeeshan, Q., Korhan, O., Asmael, M., & Safaei, B. (2020). Machine learning in predictive maintenance towards sustainable smart manufacturing in industry 4.0. *Sustainability, 12*(19), 8211.

[2] Bughin, J., Seong, J., Manyika, J., Chui, M., & Joshi, R. (2018). Notes from the AI frontier: Modeling the impact of AI on the world economy. *McKinsey Global Institute, 4*.

[3] Russell, S. J., & Norvig, P. (2022). Artificial intelligence: A modern approach (4th ed.). Pearson.

[4] Peres, R. S., Jia, X., Lee, J., Sun, K., Colombo, A. W., & Barata, J. (2020). Industrial artificial intelligence in industry 4.0-systematic review, challenges and outlook. *IEEE Access, 8,* 220121-220139.

[5] Dankwa, S., & Zheng, W. (2019). Special issue on using machine learning algorithms in the prediction of kyphosis disease: a comparative study. *Applied Sciences*, *9*(16), 3322.

[6] Theissler, A., Pérez-Velázquez, J., Kettelgerdes, M., & Elger, G. (2021). Predictive maintenance enabled by machine learning: Use cases and challenges in the automotive industry. *Reliability engineering & system safety, 215*, 107864.

[7] Zhang, C., & Lu, Y. (2021). Study on artificial intelligence: The state of the art and future prospects. *Journal of Industrial Information Integration*, *23*, 100224.

[8] Janiesch, C., Zschech, P., & Heinrich, K. (2021). Machine learning and deep learning. *Electronic Markets*, *31*(3), 685-695.

[9] Kiangala, K. S., & Wang, Z. (2020). An effective predictive maintenance framework for conveyor motors using dual time-series imaging and convolutional neural network in an industry 4.0 environment. *Ieee Access*, *8*, 121033-121049.

[10] Kiranyaz, S., Avci, O., Abdeljaber, O., Ince, T., Gabbouj, M., & Inman, D. J. (2021). 1D convolutional neural networks and applications: A survey. *Mechanical systems and signal processing*, *151*, 107398.

[11] Serradilla, O., Zugasti, E., Rodriguez, J., & Zurutuza, U. (2022). Deep learning models for predictive maintenance: a survey, comparison, challenges and prospects. *Applied Intelligence*, *52*(10), 10934-10964.

[12] Salehinejad, H., Sankar, S., Barfett, J., Colak, E., & Valaee, S. (2017). Recent advances in recurrent neural networks. *arXiv preprint arXiv:1801.01078*.

[13] Medsker, L. R., & Jain, L. C. (2001). Recurrent neural networks. *Design and Applications*, *5*, 64-67.

[14] Na Pattalung, T., Ingviya, T., & Chaichulee, S. (2021). Feature explanations in recurrent neural networks for predicting risk of mortality in intensive care patients. *Journal of Personalized Medicine*, *11*(9), 934.

[15] Zonta, T., Da Costa, C. A., da Rosa Righi, R., de Lima, M. J., da Trindade, E. S., & Li, G. P. (2020). Predictive maintenance in the Industry 4.0: A systematic literature review. *Computers & Industrial Engineering*, *150*, 106889.

[16] Mobley, R. K. (2002). *An introduction to predictive maintenance*. Elsevier.

[17] Yu, Y., Si, X., Hu, C., & Zhang, J. (2019). A review of recurrent neural networks: LSTM cells and network architectures. *Neural computation*, *31*(7), 1235-1270.

[18] Dey, D., & Jana, R. (2022, November). Bearing Fault Predictive Maintenance using LSTM. In *2022 3rd International Conference on Computing, Analytics and Networks (ICAN)* (pp. 1-6). IEEE.

[19] Gao, R., Huo, Y., Bao, S., Tang, Y., Antic, S. L., Epstein, E. S., ... & Landman, B. A. (2019). Distanced LSTM: time-distanced gates in long short-term memory models for lung cancer detection. In *Machine Learning in Medical Imaging: 10th International Workshop, MLMI 2019, Held in Conjunction with MICCAI 2019, Shenzhen, China, October 13, 2019, Proceedings 10* (pp. 310-318). Springer International Publishing.

[20] Okut, H. (2021). Deep learning for subtyping and prediction of diseases: Long-short term memory. In *Deep Learning Applications*. IntechOpen.

[21] Smagulova, K., & James, A. P. (2019). A survey on LSTM memristive neural network architectures and applications. *The European Physical Journal Special Topics*, *228*(10), 2313-2324.

[22] Li, Z., He, D., Tian, F., Chen, W., Qin, T., Wang, L., & Liu, T. (2018, July). Towards binary-valued gates for robust lstm training. In *International Conference on Machine Learning* (pp. 2995-3004). PMLR.

[23] Pulver, A., & Lyu, S. (2017, May). LSTM with working memory. In *2017 International Joint Conference on Neural Networks (IJCNN)* (pp. 845-851). IEEE.

[24] BERRAJAA, A. (2022). Natural Language Processing for the Analysis Sentiment using a LSTM Model. *International Journal of Advanced Computer Science and Applications*, *13*(5).

[25] Zhao, T. (2019, July). Deep multimodal learning: An effective method for video classification. In *2019 IEEE International Conference on Web Services (ICWS)* (pp. 398-402). IEEE.

[26] Cryer, J. D., & Kellet, N. (1991). *Time series analysis*. Royal Victorian Institute for the Blind. Tertiary Resource Service.

[27] Vennerød, C. B., Kjærran, A., & Bugge, E. S. (2021). Long short-term memory RNN. *arXiv preprint arXiv:2105.06756*.

[28] Lindemann, B., Maschler, B., Sahlab, N., & Weyrich, M. (2021). A survey on anomaly detection for technical systems using LSTM networks. *Computers in Industry*, *131*, 103498.

[29] Malhotra, P., Vig, L., Shroff, G., & Agarwal, P. (2015, April). Long Short-Term Memory Networks for Anomaly Detection in Time Series. In *ESANN* (Vol. 2015, p. 89).

[30] Bank, D., Koenigstein, N., & Giryes, R. (2020). Autoencoders. arXiv preprint arXiv:2003.05991.

[31] Saumya, S., & Singh, J. P. (2022). Spam review detection using LSTM autoencoder: an unsupervised approach. *Electronic Commerce Research*, *22*(1), 113-133.

[32] Kang, J., Kim, C. S., Kang, J. W., & Gwak, J. (2021). Anomaly detection of the brake operating unit on metro vehicles using a one-class LSTM autoencoder. *Applied Sciences*, *11*(19), 9290.

[33] Do, J. S., Kareem, A. B., & Hur, J. W. (2023). LSTM-Autoencoder for Vibration Anomaly Detection in Vertical Carousel Storage and Retrieval System (VCSRS). *Sensors*, *23*(2), 1009.

[34] Yan, Y., Qi, L., Wang, J., Lin, Y., & Chen, L. (2020, June). A network intrusion detection method based on stacked autoencoder and LSTM. In *ICC 2020-2020 IEEE International Conference on Communications (ICC)* (pp. 1-6). IEEE.

[35] Pravin, S. C., & Palanivelan, M. (2021). Regularized deep LSTM autoencoder for phonological deviation assessment. *International Journal of Pattern Recognition and Artificial Intelligence*, *35*(04), 2152002.

[36] Nguyen, H. D., Tran, K. P., Thomassey, S., & Hamad, M. (2021). Forecasting and Anomaly Detection approaches using LSTM and LSTM Autoencoder techniques with the

applications in supply chain management. *International Journal of Information Management*, *57*, 102282.

[37] Sagheer, A., & Kotb, M. (2019). Unsupervised pre-training of a deep LSTM-based stacked autoencoder for multivariate time series forecasting problems. *Scientific reports*, *9*(1), 1-16.

[38] Yuan, L. P., Liu, P., & Zhu, S. (2020). Recomposition vs. Prediction: A Novel Anomaly Detection for Discrete Events Based on Autoencoder. *arXiv preprint arXiv:2012.13972*.

[39] Bampoula, X., Siaterlis, G., Nikolakis, N., & Alexopoulos, K. (2021). A deep learning model for predictive maintenance in cyber-physical production systems using lstm autoencoders. *Sensors*, *21*(3), 972.

[40] Kamat, P., & Sugandhi, R. (2020). Anomaly detection for predictive maintenance in industry 4.0-A survey. In *E3S web of conferences* (Vol. 170, p. 02007). EDP Sciences.

[41] François-Lavet, V., Henderson, P., Islam, R., Bellemare, M. G., & Pineau, J. (2018). An introduction to deep reinforcement learning. *Foundations and Trends® in Machine Learning*, *11*(3-4), 219-354.

[42] Poór, P., Basl, J., & Zenisek, D. (2019, March). Predictive Maintenance 4.0 as next evolution step in industrial maintenance development. In *2019 International Research Conference on Smart Computing and Systems Engineering (SCSE)* (pp. 245-253). IEEE.

[43] Coandă, P., Avram, M., & Constantin, V. (2020, December). A state of the art of predictive maintenance techniques. In *IOP Conference Series: Materials Science and Engineering* (Vol. 997, No. 1, p. 012039). IOP Publishing.

[44] Ran, Y., Zhou, X., Lin, P., Wen, Y., & Deng, R. (2019). A survey of predictive maintenance: Systems, purposes and approaches. *arXiv preprint arXiv:1912.07383*.

[45] O'Donoghue, B., Munos, R., Kavukcuoglu, K., & Mnih, V. (2016). Combining policy gradient and Q-learning. *arXiv preprint arXiv:1611.01626*.

[46] Han, M., Zhang, L., Wang, J., & Pan, W. (2020). Actor-critic reinforcement learning for control with stability guarantee. *IEEE Robotics and Automation Letters*, *5*(4), 6217-6224.

[47] Zhou, Z. H. (2018). A brief introduction to weakly supervised learning. *National science review*, *5*(1), 44-53.

[48] Dike, H. U., Zhou, Y., Deveerasetty, K. K., & Wu, Q. (2018, October). Unsupervised learning based on artificial neural network: A review. In *2018 IEEE International Conference on Cyborg and Bionic Systems (CBS)* (pp. 322-327). IEEE.

[49] Van Engelen, J. E., & Hoos, H. H. (2020). A survey on semi-supervised learning. *Machine learning*, *109*(2), 373-440.

[50] Dong, S., Wang, P., & Abbas, K. (2021). A survey on deep learning and its applications. *Computer Science Review*, *40*, 100379.

[51] Yasrab, R., Pound, M. P., French, A. P., & Pridmore, T. P. (2020). PhenomNet: bridging phenotype-genotype gap: a CNN-LSTM based automatic plant root anatomization system. *bioRxiv*, 2020-05.

[52] Gabriel Loye 2019, Long Short-Term Memory: From Zero to Hero with PyTorch. In FloydHub Blog.

[53] Sublime, J. (2021). *Contributions to modern unsupervised learning: Case studies of multi-view clustering and unsupervised Deep Learning.*

[54] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735-1780.

[55] Ribeiro, F.M.L.: *MaFaulDa-Machinery Fault Database* Signal, Multimedia and Telecommunications Laboratory, federal university of Rio De Janeiro.