POLITECNICO DI TORINO

College of Computer Engineering, Cinema and Mechatronics

Master's Degree Thesis

A novel framework for condition-based maintenance and performance analysis based on data-driven approaches

Supervisors
prof. Bartolomeo Montrucchio
dr. Antonio Costantino Marceddu

Candidate
Francesco Cartelli

April 2023

Summary

Over the past decade, wind energy has become increasingly significant in the global energy sector. Nonetheless, *operation and maintenance* (O&M) account for at least one third of the overall energy generation cost. *Condition-based maintenance* (CBM) provides a remedy for this issue: instead of scheduling maintenance, this technique monitors turbine components and performs maintenance only when certain warnings are provided (possibly anticipating any faults).
Besides the numerous studies of wind turbine generators (WTGs) on fault detection and diagnosis, all of these strategies could be categorized as model-based approaches and data-driven approaches. Model-based techniques rely mostly on a precise mathematical model of the WTG and its subsystems. In contrast, data-driven systems do not require physical or exact mathematical models, but instead infer the defect detection system from observed sensor data. The latter techniques have shown to be particularly successful in recent years for modeling complex interactions associated to wind turbines [1]. Yet, the existence of various nonlinearities in the examined issues and measurement noise necessitates the adoption of complex and robust algorithms.
This thesis proposes a framework for data-driven condition-based maintenance: the objective of this work is to develop anomaly and failure detection algorithms, that can be later used to provide maintenance on condition.
To this end, an *unsupervised learning* method, involving several *auto-encoder* (AE) neural network models (FNN as well as RNN) is provided.
The dataset of this work comes from real world *SCADA* measurements of Sirius s.r.l., a partner of important companies in the world of renewable energies. The turbines data, belonging to various plants located in southern Italy, is collected from mechanical and thermic sensors measurements, every 10 minutes. The considered problem includes also different turbine designs, with distinct engineering designs.
In the process, data from SCADA systems is acquired and clustered based on WT performances, relying on *key performance indicators*, *turbines statuses* and *alarms*. The best-performing time sequences are then selected as inputs for the subsequent training phase.
In an unsupervised learning manner, different AE models are trained in a multivariate time series reconstruction task. During this phase, the models learn a robust latent representation of the time series key features.
When used on unseen data, the algorithm will reconstruct the provided input sequences, the reconstruction error is then analyzed for anomalies detection. in this study the different autoencoder models will then be exposed to different
regularization approaches such as *dropout* and *de-noise autoencoder* (DAE) assessing the different robustness of the models produced. The various AE architectures are then tested in a simulated benchmark environment, in which anomalies, noises and faulty behaviors are injected in order to be detected. The most promising models are then employed in a real world test-case, in which, previously labelled WTG critical events need to be detected.
The objective of the analysis will not only be to detect adverse events, but also to correctly identify the measures and subsystems implicated in the anomalies.
With this study, the efficacy of data-driven AI-powered ways to acquire evaluations on the existence and nature of anomalies has been demonstrated, for the possibility of these models in generating nonlinear relationships by effectively simulating real-world contexts.. Also, the efficacy of the method permits an evaluation of the performance of wind farms and their subsystems.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Wind power is a source of renewable energy that is seeing increasing adoption while also having significant global impact. The generation of energy via the use of wind power has a number of positive effects, including phase out fossil fuels, and encouraging sustainable development.

During the last several years, there has been a substantial rise in the use of wind energy, and there are currently wind turbines constructed in more than 90 nations throughout the globe. According to the Global Wind Energy Council, there is enough wind energy capacity built throughout the globe to produce over 700 gigawatt, which accounts for around 7% of the total power produced worldwide. Wind energy plays an even more major role in some nations, such as Denmark, which generates more than fifty percent of its power from wind energy.

It is also notable that wind energy may be used in a variety of contexts, ranging from large-scale wind WPP to small-scale home systems. This adaptability allows wind energy to be utilized in a variety of situations. Wind energy may be especially useful in outlying places since it has the potential to serve as a dependable and inexpensive source of electrical power in these locations. Moreover, developments in wind turbine technology have made it feasible to collect electricity from low-wind-speed places as well as offshore sites, which has further expanded the possibilities for the use of wind energy. The use of wind energy, despite the numerous advantages it offers, is not without its share of difficulties. Intermittency is one of the key obstacles, since wind energy generation is reliant on the availability of wind. This makes it one of the primary challenges. This may lead to difficulties in the integration and stability of the grid, which calls for rapid and affordable solutions such as energy storage and demand response.

Although the use of wind energy has the potential to play a significant role in the global transition to a more sustainable and low-carbon energy system, the costs of maintenance and repair are a significant part of the overall cost of wind energy (maintenance and repair costs can account for 20% to 25% of the total operating costs of a wind power plant and can cause a reduction in energy production of up to 5%). This is despite the fact that wind energy has the potential to play a significant role in this transition. Nevertheless, recent developments in condition monitoring and predictive maintenance technology have helped to bring these expenses down while simultaneously increasing the overall reliability and availability of the wind turbines.

Wind turbine operators are able to spot possible defects and maintenance requirements before they result in expensive downtime or equipment breakdowns if they use appropriate condition monitoring and maintenance techniques. This not only lowers the costs of maintenance and replacement, but it also increases the lifetime of wind turbines, which leads to an improvement in the economic performance of the turbines as a whole. Therefore, despite the fact that the costs of maintenance and repair are a significant factor in the adoption and expansion of wind energy, the ongoing development of technologies for condition monitoring and predictive maintenance, along with improvements in wind turbine design, will help to ensure that wind energy continues to be an affordable and sustainable source of renewable energy.

1.1 Scope of the Thesis

The primary objective of this thesis is to provide the groundwork for the development of a condition-based maintenance algorithm for wind turbine generators (WTG). This algorithm will be developed to identify weak performance and downtimes in WTG and offer measurements that correspond to the system constituent parts.

In order to accomplish this objective, our primary emphasis will be on the development of algorithms that are in a position to recognize and identify periods of poor performance and downtime in an anomaly detection task. The system will be able to spot trends and abnormalities that flag possible performance difficulties by evaluating data from numerous sources, including sensors and other monitoring devices. This data will be analyzed by our algorithm. The capability of the framework to tie poor performance to certain subcomponents of the WTG system is one of the most important aspects of the proposed methods. In this way it is possible to discover the underlying causes of performance difficulties and build maintenance methods that are specifically tailored to solve them.

The creation of a condition-based maintenance algorithm for WTG, in general, may have substantial repercussions for the wind energy sector. This algorithm has the potential to further reduce the cost and encourage its broad adoption by enhancing the efficiency of maintenance and increasing reliability.

1.2 Background and Solution

The study makes use of a dataset that is the intellectual property of Sirius s.r.l., and it is comprised of a number of wind power plants (WPPs) that are situated in southern Italy. Observation of SCADA data originating from customer WPPs is made possible via the apps that Sirius has created and is offering to customers. The 10-minutes average statistics that are provided here give a collection of magnitudes that were acquired in real time and aggregated.

In the proposed problem, only one wind farm will be considered in the experiment. However, within this park different types of turbines, with different geometrical and mechanical characteristics, will be considered. Together with the associated measurements, each turbine allows for a set of indicators to observe the operating status of these turbines and even verify the malfunction of their subsystems by integrating an alarm system.
While these technologies make it possible to identify particular faults of a variety of system components, the alerts generated by these tools are not adequate to completely capture all of the potential irregularities and performance decreases that may occur. The number of these events then constitutes an extremely small amount (less than a dozen events for each subsystem per year each lasting around 10 minutes), so learning about these malfunctions cannot be based on mere observation of them.
For this reason, the algorithm exploits an unsupervised type of anomaly detection. Not being able to explicitly isolate and study these down moments, the problem is turned on its head, instead developing an algorithm that learns to obtain a representation of optimal turbine operation. This algorithm, working with appropriately filtered data, according to logic related to performance and elimination of malfunction states, should autonomously learn to recognize certain moments and, much more importantly, report malfunctions and poor performance.

The autoencoder (AE) is a sort of neural network that was developed expressly for situations like these. It was meant to solve them quickly and accurately. In point of fact, AEs are networks that lend themselves very well to the process of learning a condensed representation of a dataset. A representation that will thereafter be utilized to carry out a job in response to the inputs that have been received. The outcome of this mission will be included into a discrimination method that will be used to differentiate between the proper actions of the turbine and its faults. After that, this outcome may be put through the assessments that are inherent in its distribution, which will subsequently make it possible to construct rankings and histories of the functioning of these different systems.

The activity that will be performed by these AE will be a straightforward data reconstruction activity. As an input to the network, the time sequences will be compressed and rebuilt while attempting to reduce the amount of reconstruction error that occurs. If the AE was trained on data that was properly reflective of good performance, then its results will be outstanding when using comparable data, but they will be insufficiently accurate when using data from other sources.

In a series of tasks involving anomaly detection, various types of AE will be used and evaluated. These tasks will assess the capacity of the AE to distinguish between good data and irregular data, as well as their ability to obtain an accurate interpretation of anomalous phenomena and correlate it to various subsystems. The different tasks to be solved will be:

- Ability to recognize test data filtered with the same techniques and metrics as training data.
- Ability to recognize error and idle status of turbines.
- Ability to recognize anomalies in artificially introduced data.
- Adequately explain the different measures involved in a well-documented anomaly both quantitatively and qualitatively.

In the end, the algorithm will be implemented in the process of comprehending and assessing a group of turbines that are part of the wind farm. In addition to that, this will give a use case for the method in the context of maintenance.

Chapter 2

General Notions

2.1 Wind Energy and Wind Turbine Generators Principles

Wind energy is the energy that can be extracted from the movement of air masses, generally called wind. Those air masses are moved due to changes in atmospheric pressure, temperature, and the rotation of the Earth.

2.1.1 Wind and its Energy

The uneven heating of the surface of the Earth by the sun causes various regions to have varying temperatures. This temperature difference creates air pressure changes. Since heated air is less dense than cool air, it rises, while cool air descends. Wind is caused by the passage of air from places of high pressure to areas of low pressure.
In addition to temperature variations, the planet rotation also influences the movement of air masses. The Coriolis effect, caused by the rotation of the Earth, causes the wind to deflect to the right in the Northern Hemisphere and to the left in the Southern Hemisphere. This deviation causes the wind to bend rather than travel in a straight line.
Additional variables, including the topography of the land and the existence of significant bodies of water, can also influence wind patterns. For instance, mountains may force the wind to flow around them, and big bodies of water can cause the wind direction to alter when it interacts with the water surface.
The fundamental attraction for coastal or marine wind farms is that wind blows faster over vast, flat areas like the sea. Wind speeds up on plateaus or valleys parallel to the primary wind direction and slows down on uneven surfaces like cities or woods.
Overall, the movement of air masses is a dynamic and complicated process that is influenced by a number of elements. Wind energy provides a sustainable and clean source of electricity by capturing the energy contained in this movement.
When managing wind-generated energy, it is crucial to account for the vast changes in wind velocity across different locations: sites merely a few kilometers apart may be exposed to different circumstances and have distinct interests for the construction of wind turbines [2].

2.1.2 Wind Turbines

Wind turbines are machines that convert wind energy to electricity. Blades are coupled to a rotor that is connected to a generator placed in a tower. The blades collect the kinetic energy of the wind, causing the rotor to revolve and the generator to produce electricity. Wind turbines can be utilized as independent devices or linked to the power grid to provide communities with renewable energy [3].

2.1.3 Wind Turbines Components

These are the main components of a generic wind turbine generator [4] (Figure 2.1):

- The **tower** of a wind turbine is the structure that supports the rotor and generator. It is normally constructed from steel or concrete and is tall enough to take advantage of the greater and more constant wind speeds at higher elevations. The tower height is crucial to the overall efficiency and performance of the wind turbine, as the quantity of accessible wind energy grows with height. In addition, the tower houses the gearbox, which increases the rotor rotational speed to create power, and the electrical components that convert mechanical energy into electrical energy. The tower may be set on a concrete base or fixed with guy wires.

- A wind turbine **blade** is a long, thin structure linked to a wind turbine rotor that transforms the kinetic energy of the wind into rotational energy. Blades of wind turbines are often constructed from composite materials, such as fiberglass or carbon fiber, and are designed to be aerodynamic and effective at gathering wind energy. The blade form is tailored to maximize the amount of energy collected while reducing resistance and structural stress. The blades are fixed on a hub that is attached to the rotor shaft and are normally oriented with their leading edge pointing into the wind. The blades revolve with the wind, causing the rotor to rotate and generating electricity through the tower generator. The total performance and efficiency of a wind turbine is significantly impacted by the blade dimensions and designs [5]. The majority of contemporary wind turbines feature three or two blades. Wind turbines with three blades are the most prevalent and commonly utilized due to their efficiency, dependability, and ease of maintenance. The balanced and symmetrical design of three-bladed turbines helps to decrease structural stresses and increase the rotor stability. Two-bladed wind turbines are also used due to their light weight and simplicity compared to their three blades counterparts.

- The **gearbox** of a wind turbine is a mechanical system that increases the rotor rotating speed. The gearbox is positioned in the tower and is connected to the generator and rotor respectively by a *high-speed shaft* and a *low-speed shaft*. The objective of the gearbox is to convert the sluggish spinning speed of the rotor into a sufficient speed for power generation. The gearbox normally employs a set of gears to raise the rotating speed by a factor of 50 or more, enabling the generator to produce alternating current (AC) with a higher frequency suited for the power grid. The gearbox is an essential component of the wind turbine because it permits the conversion of wind kinetic energy to electrical energy. Nonetheless, it is a high-wear component that is subject to tremendous stress and must be routinely serviced to preserve the wind turbine durability and dependability.

- The **rotor** of a wind turbine is the component that transforms the kinetic energy of the wind into rotational energy. Typically, the rotor consists of many blades joined to a hub that is coupled to a shaft. When wind blows over the blades, lift is created, causing the rotor to revolve. The rotor is linked to a generator within the tower, and the rotor spinning causes the generator to generate power. The size, number, and diameter of the rotor blades, as well as the rotor diameter, are crucial design elements that influence the overall performance and efficiency of the wind turbine. Typically, the rotor is placed towards the top of the tower, where it can take advantage of the greater and more constant wind speeds. The rotor revolves on a horizontal axis, and when viewed from above, the direction of rotation is generally clockwise. Vertical wind turbines with big cup- or curved-shaped blades utilize drag designs more frequently. The wind physically moves the blades, which are coupled to a central shaft, out of the way. The slower rotational speeds and strong torque capabilities of drag-designed rotor blades make them suitable for water pumping and agricultural machinery power. Lift-powered wind turbines have a far faster rotating speed than drag-powered turbines and are thus ideal for energy generation.

- A wind turbine **nacelle** is a housing that encloses the wind turbines primary components, such as the rotor, gearbox, generator, and control systems. Typically, the nacelle is positioned at the top of the tower and is situated behind the rotor blades. It is often built

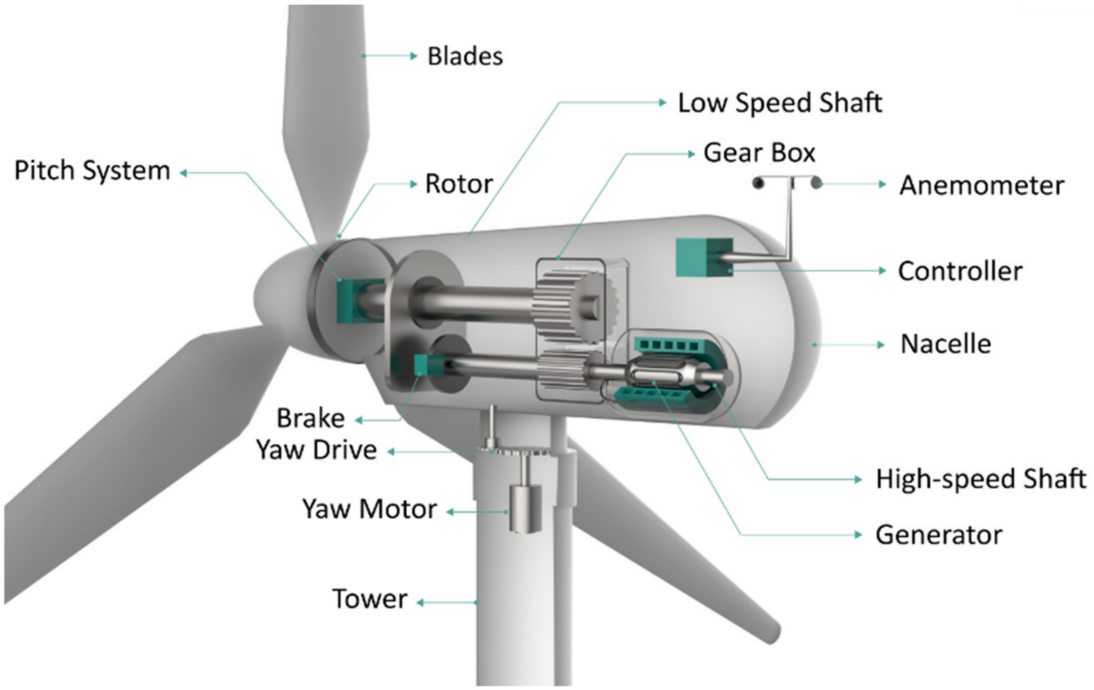


Figure 2.1: Wind turbine generator scheme. Image from [6].

of fiberglass or other composite materials and is intended to shield the inside components from wind, rain, and snow. The nacelle also features maintenance and inspection access ports and hatches. Depending on the design of the wind turbine, the size and shape of the nacelle will vary, although it will normally be streamlined to reduce drag and aerodynamic resistance. The nacelle is an essential component of the wind turbine, since it contains and protects the electricity-generating components.

- A wind turbine **generator** turns the kinetic energy of a wind turbine rotor into electrical energy. Typically, the generator is housed in the tower and is linked to the rotor by a shaft and gearbox. As the rotor revolves, it drives the generator to provide an output of alternating current (AC). Depending on the wind turbine design, the generator may be a permanent magnet generator, an induction generator, or a synchronous generator.
  The type of generator utilized influences the wind turbine efficiency, cost, and control system complexity. The generator is an essential component of the wind turbine, since it generates the electrical energy that may be consumed directly or fed into the power grid. Typically, the generator is cooled by air or a dedicated cooling system, and it may be equipped with a number of sensors and controls to monitor its operation and prevent overloading or failure.

- **Control system** of electronics, sensors, and software that monitors and controls the wind turbine functioning. The control system is responsible for starting and halting the wind turbine, regulating the blade pitch, and monitoring the generator and other components performance.

- **Yaw Drive** is a device that spins the nacelle and rotor in order to align them with the direction of the wind. Typically, the yaw drive is located near the tower's base and is powered by an electric motor.

- **Anemometers** are instruments that measure the speed and direction of the wind. The anemometer is utilized to regulate the yaw drive and enhance the rotor's performance in the wind by adjusting the rotor's location.

- A system of mechanical or hydraulic **brakes** that may be used to halt the rotor's spinning in an emergency or during maintenance.

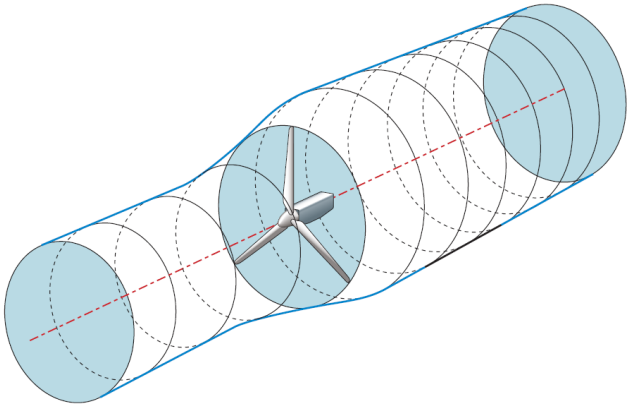


Figure 2.2: Flow pattern around a wind turbine.

2.1.4 Wind Turbines Energy Model

The power associated with the flow (Figure 2.2) of a moving air mass is equal to:

$$P = \frac{1}{2}\rho S_r V_0^3$$

where ρ is the air density, S_r is the section through which the air mass flows, and V_0 is the wind speed. The rotor faces the wind, so the S_r represents the area of the rotor. Since a certain amount of energy is subtracted from the wind kinetic energy, the speed downwind of the rotor results is lower than the upwind one. As a result, the diameter of the flow tube at the back of the rotor is greater than that at the front. This behavior is depicted in Figure 2.2.

Thanks to A. Betz, it was discovered that it is impossible to entirely convert the kinetic energy of a mass of air into mechanical energy. He also discovered that there is an upper limit to the amount of kinetic energy that can be converted. Due to this reason, Betz introduced a parameter, called the power coefficient C_p, which can be calculated as a function of the ratio of the speed of the wind wake behind the rotor to the speed ahead. The power produced by a wind turbine is thus [7]:

$$P_{wtg} = \frac{1}{2}C_p\rho S_r V_0^3$$

The optimal value of this parameter is 0.593 [8]. As a result, convertible energy accounts for roughly one-third of wind energy. Therefore, it is not possible to design a turbine with a higher value of the power coefficient. But, this condition occurs for a rotor under ideal conditions when the outgoing air velocity is one-third of the incoming air velocity and the rotor is infinitely thin. Today's turbines have power coefficients of about 70-80% of the theoretical limit. On a WTG, there are two primary controls:

- **Blade pitch angle control**: a blade is similar to a wing. The surface area available to the incoming wind is critical for increasing aerodynamic forces on the rotor blades. The angle of attack α is defined as the angle between the incident flow vector and the plane of the blade segment, while the pitch angle β represents the angle between the plane of the blade segment and the plane of the rotor. Blade pitch angle control can rotate the blades around their axis in such a way as to have, for each wind speed, an optimal blade angle of attack (Figure 2.3).
- **Yaw control**: by rotating the nacelle, the turbine is oriented, actively or passively, in the direction of the wind to maximize the efficiency of the energy conversion process (Figure 2.3).

Both types of controls are depicted in Figure 2.3.

The rotor blades utilize either the lift or drag principle to capture energy from air masses in motion. The design of the lift blades is based on the same concept that allows airplanes, kites, and birds to fly, providing a lifting force perpendicular to the direction of motion. Essentially, the rotor blade is an aerofoil, or wing comparable in form to an airplane wing. As the blade slices through the air, a disparity in wind speed and pressure is produced between the upper and bottom sides of the blade. The stronger pressure at the bottom surface "lifts" the blade upward, by design this force has to be as great as possible. This lift is transformed into a rotating motion when the blades are joined to a central rotational axis, as in a wind turbine rotor. This lifting force is opposed by a drag force that is perpendicular to the direction of motion and generates turbulence around the trailing edge of the blade as it slashes through the air. This turbulence has a braking effect on the blade, thus it has to be minimized. The combination of lift and drag generates the rotor's propeller-like rotation.

The rotor blades revolve around a center bearing, making a complete 360-degree circle. Therefore, as the swept area of the rotor rises, the area it covers increases proportionally with the radius squared. Therefore, doubling the length of a turbine's blades results in a quadrupling of its surface area, allowing it to capture four times as much wind energy. However, this significantly increases the wind turbine's size, weight, and eventually cost. The rotational tip-speed of the rotor resulting from the angular velocity is a significant aspect of the blade length. The greater the length of the turbine blade, the faster the spinning of the tip at a given wind velocity. Similarly, for a given length of rotor blades, the greater the wind speed, the faster the spin. Why then can't a wind turbine design have extremely long rotor blades that operates in a windy area and generates a great deal of free power from the wind? The explanation is that there is a point at which the length of the rotor blades and the wind speed diminish the output efficiency of the wind turbine. This is why many designs for larger wind turbines revolve at significantly slower rates. Efficiency is a function of how quickly the rotor tip rotates at a given wind speed, generating a constant wind speed to tip ratio known as the "tip-speed ratio", which is a dimensionless unit used to maximize rotor efficiency. In other terms, "tip-speed ratio" (TSR) is the ratio of the revolving blade tip speed in revolutions per minute (rpm) to the wind speed in kilometers per hour (Kph) or miles per hour (mph) (mph).

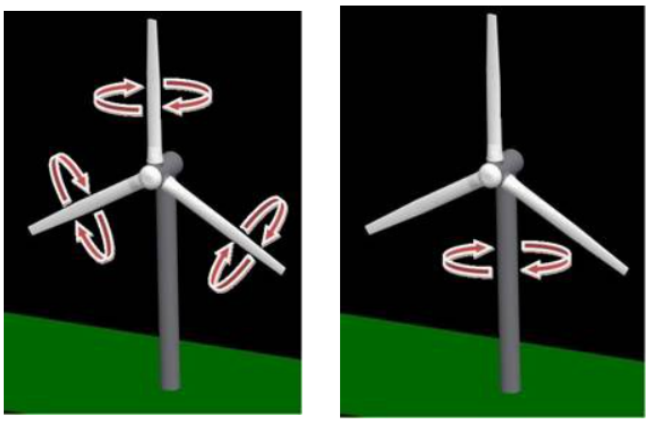


Figure 2.3: From left to right: blade pitch angle and yaw controls.

2.1.5 Power Curve

A power curve is a graphical representation of the relationship between wind speed and the power output of a wind turbine. It shows how the power output of a wind turbine changes with changes in wind speed. Its shape is unique to each wind turbine and is determined by several factors [9], including the size and design of the blades, the height of the tower, and the efficiency of the generator.
The power curve can be used to determine the wind speeds at which the wind turbine will operate most efficiently and generate the maximum power output. It can also be used to identify any operational limitations of the wind turbine, such as *cut-in* and *cut-out* speeds, and to optimize the performance of the wind turbine by adjusting its operating parameters.

The shape of the power curve can vary depending on the type of wind turbine and its operating conditions [10], but typically shows an exponential increase in power output with increasing wind speed up to a peak, after which the power output levels off and eventually decreases with higher wind speeds. In general, wind turbines are designed to generate maximum power at wind speeds between 8 and 25 meters per second (m/s). Typically, the power curve of a wind turbine consists of three regions: the *cut-in region*, the *rated region*, and the *cut-out region* (Figure 2.4).
Each region is limited by an homonymous speed, those speeds are described below.

Cut-in-speed

The cut-in speed is the wind speed at which the turbine starts to generate power.
When the wind speed is below this threshold, the wind turbine's blades cannot revolve quickly enough to produce power. After the wind speed reaches the cut-in speed, the blades of the turbine will begin to revolve and produce electricity. Up to a certain point, the quantity of electricity generated by the turbine grows as the wind speed increases.

Rated output speed

The rated output speed is the rotational speed at which a WTG generates its maximum rated output power.
Typically, the rated output speed is determined by the turbine manufacturer based on the design and performance parameters of the turbine. It is a crucial characteristic for wind turbines since it controls the speed at which the generator achieves its maximum power output and, therefore, the quantity of electricity the turbine can produce. Typically, the rated output speed is less than the maximum speed that the turbine's blades can achieve. This is due to the fact that the generator is intended to function most efficiently within a specified speed range, and the turbine management system is designed to keep the turbine's speed within this range. By running at its rated output speed, the generator may produce the most power while incurring the least amount of component wear and tear.

Cut-out-speed

The cut-out speed is the wind speed at which a wind turbine automatically shuts down to prevent damage.
Wind turbines have a maximum working speed, which corresponds to the speed at which they generate the most electricity. Nevertheless, when wind speeds surpass this maximum working speed, the turbine's blades might spin too quickly, resulting in significant wear and tear on the generator and gearbox. This may result in damage and, in severe situations, complete turbine failure. A cut-out speed is chosen to prevent damage to the turbine. When the wind speed reaches the cut-out speed, the control system of the turbine activates a device that slows and shuts down the turbine to prevent damage. Typically, the cut-out speed is set above the maximum working speed to give a safety buffer and guarantee that the turbine shuts down prior to being exposed to damaging wind speeds.

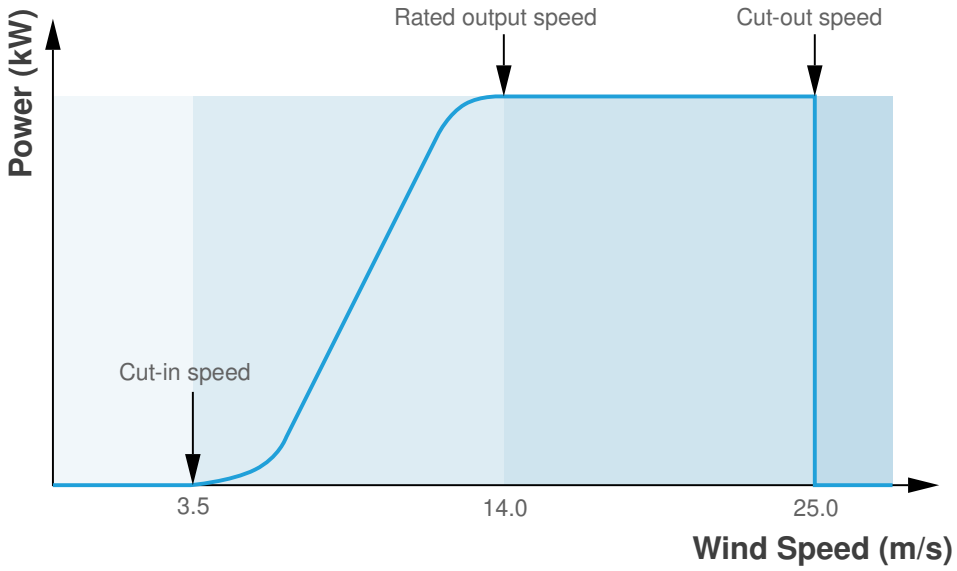


Figure 2.4: Ideal power curve, showing the different regions and speeds.

2.2 Condition-Based Maintenance

Condition-Based Maintenance, often known as CBM, is a technique for preventative maintenance that determines when maintenance should be conducted by monitoring the condition of the equipment and systems in real time. It makes it possible to plan maintenance depending on the current conditions, rather than planned maintenance (PM). This last strategy is conducted at predetermined intervals, while CBM is performed only when a decline in the equipment's condition is recognized [11]. This extends the time between maintenance fixes compared to preventative maintenance, as maintenance is performed when needed.

To be successful, CBM operation, must also include a number of additional components. This involves establishing a system for scheduled maintenance that allows to examine equipment, identify abnormalities, and trigger prompt follow-up repair orders. This is an advantage over the alternatives. This strategy may result in a more effective use of resources as well as lower expenses associated with upkeep. CBM gathers data and provides information on the status of equipment through the use of a variety of technologies and procedures [12], including as vibration analysis, thermal imaging, and oil analysis.
There are a few different approaches of data collection that may be used for CBM:

- **Vibration Analysis** refers to the monitoring of the vibration levels of machinery and using software to analyze the data in order to look for any odd patterns or changes that may signal the presence of a possible issue.
- **Thermal imaging** is a technique that involves the use of thermal cameras to locate and quantify heat patterns on various pieces of equipment and systems. These heat patterns can serve as an indicator of regions of increased wear or probable failure.
- In the process of **oil analysis**, samples of oil are obtained from different pieces of equipment and analyzed in order to look for evidence of wear and contamination while also determining the state of the lubricating oil.
- **Ultrasonic testing** refers to a technique that employs sound waves with a high frequency in order to examine the state of the apparatus being tested and locate any potential issues.
- **Acoustic testing** can be used to detect gas, liquid or vacuum leaks.
- **Software for Predictive Maintenance** are used to evaluate and interpret data from sensors and other monitoring equipment in order to forecast when maintenance will be

necessary. These are some of the most prevalent approaches that are taken in the process of collecting data for CBM. The approach that is used is determined by the specific needs for the piece or the type of equipment that is being monitored.

2.2.1 Predictive Maintenance Software

The software known as "Predictive Maintenance" is a subcategory of CBM software. It is designed to assist in determining when components of a system or piece of equipment are likely to fail by performing real-time monitoring and analysis of data collected from sensors and other monitoring devices. The program uses algorithms and machine learning to do an in-depth analysis of the data collected on the operation of the equipment, how it is being used, and the surrounding environment in order to predict and prevent any issues. This enables maintenance teams to proactively address possible issues and plan maintenance at appropriate periods, which in turn reduces the chance of unexpected downtime and improves the reliability of the equipment [13].
Software that does predictive maintenance can also assist in the optimization of maintenance procedures, the reduction of maintenance costs, and the improvement of overall equipment performance and efficiency.

2.2.2 Key Performance Indicators

Key Performance Indicators (KPIs) are the critical (key) measurable indications of progress toward a desired outcome. KPIs serve as a focal point for strategic and operational improvement, give an analytical basis for decision making, and assist in focusing attention on what is most important [14].
In CBM, KPIs are essential for measuring the efficacy and efficiency of the maintenance approach. They give a quantifiable measurement of the CBM program's performance and can aid in identifying areas for enhancement.
KPIs are a set of metrics that organizations use to measure progress toward specific goals. In the context of CBM, they can be used to measure equipment reliability, downtime, maintenance costs, and other performance-related factors.

In WTGs, KPIs are essential for measuring the effectiveness and efficiency of the maintenance strategy [15]. They give a quantitative measurement of the WTG's performance and can assist in identifying areas for improvement.
Among the essential KPIs for WTGs are [16]:

- **Availability**: It measures the percentage of time that a WTG is available to produce power. A higher availability means that the WTG is producing more energy and generating more revenue.
- **Capacity factor**: It measures the percentage of energy that a WTG produces compared to its rated capacity. A higher capacity factor means that the WTG is producing more energy and generating more revenue.
- **Mean Time Between Failures (MTBF)**: It measures the average time between failures of the WTG. A higher MTBF means that the WTG is more reliable.
- **Mean Time to Repair (MTTR)**: It measures the average time it takes to repair a WTG after a failure. A lower MTTR means that the WTG can be returned to operation more quickly, which increases availability.
- **Energy production**: It measures the amount of energy produced by the WTG over a period of time. A higher energy production means that the WTG is producing more energy and generating more revenue.
- **Wind speed**: It measures the average wind speed at the site. A higher wind speed means that the WTG can produce more energy.

- **Turbine efficiency**: It measures the efficiency of the WTG in converting wind energy into electrical energy. A higher turbine efficiency means that the WTG is producing more energy.

These KPIs can be calculated by collecting and analyzing data pertaining to the WTG's performance over time. For instance, availability can be determined by dividing the entire operating time by the sum of the working time and the downtime. Likewise, the capacity factor can be determined by dividing the total energy produced by the WTG's rated capacity.
Overall, KPIs are an essential aspect of WTG maintenance since they enable firms to track success and identify areas for development. By monitoring KPIs on a regular basis, enterprises can guarantee that their WTGs are functioning at maximum capacity and earning the most income feasible.

2.3 Time Series Theory

Time series is a statistical method for analyzing data gathered at regular periods across time. Time series data is a set of successive observations that may be used to model and predict trends and patterns across time.
Finance, economics, meteorology, and engineering often use time series data to comprehend and anticipate the behavior of a variable over time [17]. Examples of time series data that may be used to anticipate future trends include stock prices, weather patterns, and sales information. The study of time series permits the identification of patterns and trends in data across time, as well as seasonal or cyclical fluctuations, outliers, and other abnormalities. This data may be used to create models that can predict the future values of the variable under study.
Real-world time series data can be challenging to work with due to several factors that can complicate analysis and prediction [18]. Here are some of the main problems with real-world time series data and some related terms:

- **Non-stationarity**: Many real-world time series are non-stationary, meaning that their statistical properties change over time. This can make it difficult to apply traditional time series analysis techniques, which assume that the statistical properties of the data are constant over time.
- **Missing data**: Time series data can often contain missing values, which can create gaps in the data and make it difficult to model and predict.
- **Outliers**: Time series data can also contain outliers, or extreme values that are far outside the normal range of the data. These outliers can distort statistical analyses and make it difficult to accurately model and predict future values.
- **Seasonality**: Many time series exhibit seasonality, or patterns that repeat over fixed periods of time (e.g. daily, weekly, or yearly cycles). Seasonality can complicate analysis and prediction by introducing complex patterns that must be accounted for in the models.
- **Noise**: Time series data can also contain noise, or random fluctuations that are unrelated to the underlying process being measured. Noise can make it difficult to identify meaningful patterns in the data and can lead to overfitting of models.

Real-time series can be decomposed into different components by separating time series data into its constituent factors, which can aid in identifying the data's underlying patterns and correlations. A time series is decomposed by separating it into three main components:

- **Trend**: The trend component of a time series represents the long-term direction or pattern of change in the data. It can be upward, downward, or flat, and it reflects the underlying behavior of the series over time.
- **Seasonality**: The seasonality component of a time series represents the regular and predictable fluctuations that occur within a year or over a shorter period. It reflects the systematic changes that occur due to external factors such as weather, holidays, or cultural events.

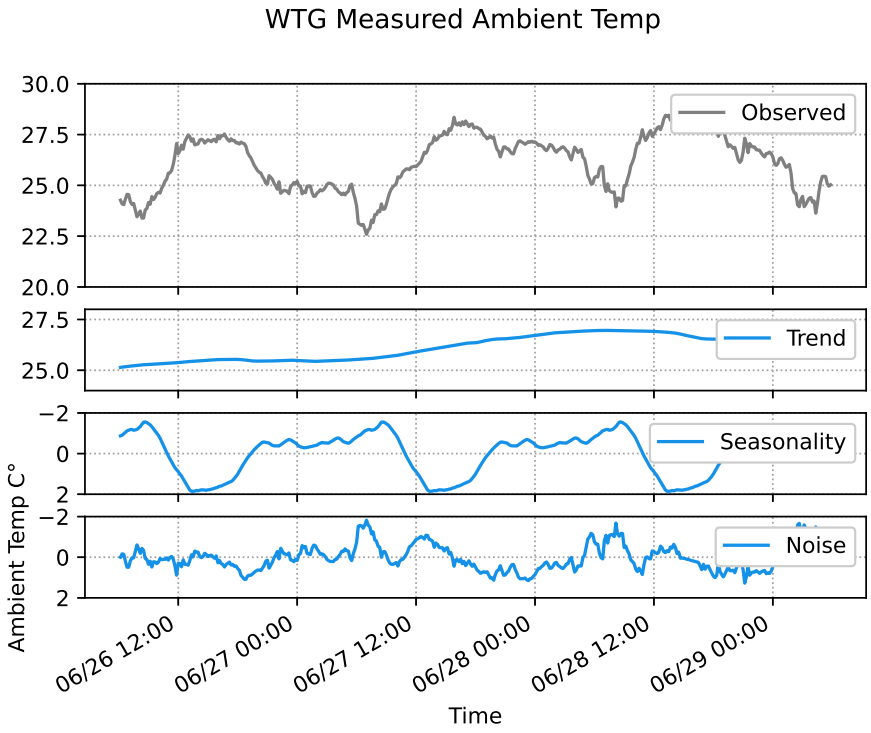


Figure 2.5: Three days of data for the ambient temperature measured by a WTG, separated in trend, seasonality and noise components. The time series appear to be periodic in each day with a slight increase in the overall temperature. Trend is computed with a 1 day centered moving average. Seasonality is obtained by considering the average measure at each timestamp in each day.

- **Noise**: The noise component of a time series represents the random or irregular fluctuations that cannot be attributed to any known factors. It reflects the unpredictable or unexplained variations in the data.

By identifying and modeling these components (an example of which is reported in Figure 2.5), a deeper understanding of the behavior of the time series can be understanded and provide more precise forecasts or predictions. Using the trend and seasonality components, for instance, a forecasting model that accounts for the underlying patterns of the data can be created, while filtering out the noise component helps to limit the impact of random fluctuations on the forecast.

2.3.1 Moving Average

In time series analysis, a moving average is a technique for smoothing out fluctuations in the data by calculating the average of a fixed number of consecutive data points. The resultant numbers indicate a moving or rolling average of the original data.

There are two primary moving average types:

- **Simple Moving Average** (SMA) is generated by averaging a specified number of consecutive data points. A 5-period SMA, for instance, would be the average of the past 5 data points. Once new data points become available, the SMA is recalculated by moving the data point window forward and including the most recent data point. An example is reported in Figure 2.6.
- **Weighted Moving Average** (WMA) gives more weight or emphasis to specific data points. Using a weighted average technique that gives more weight to more recent data points and less weight to older data points may accomplish this.

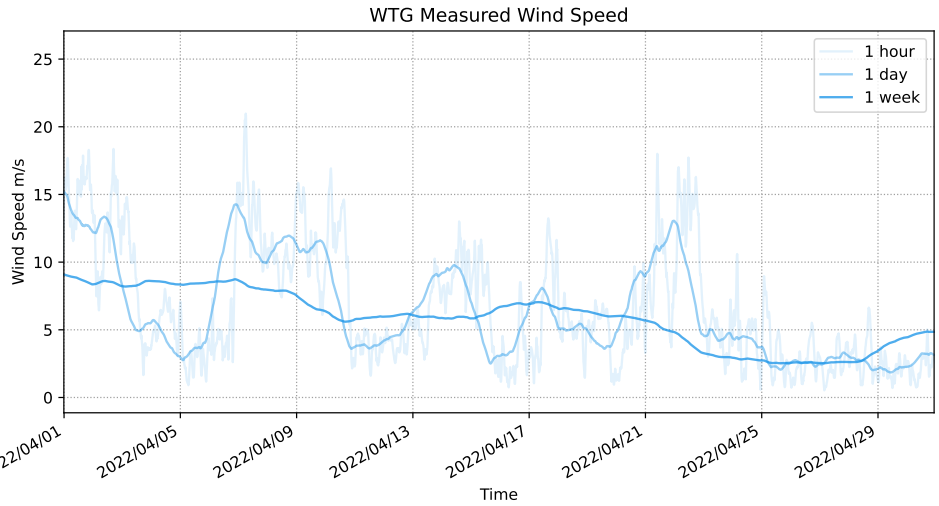


Figure 2.6: One month of data for the wind speed measured by a WTG. Each plot shows the moving average with a specific time window length obtained from 10 minutes time series data. The overlap of the three time series demonstrates the smoothing effect, which progressively remove the oscillations as the time window increases.

Moving averages are commonly used in time series analysis for several purposes:

- **Smoothing**: Moving averages can help to smooth out fluctuations in the data and highlight underlying trends or patterns.
- **Forecasting**: Moving averages can be used to forecast future values of the time series by extrapolating the trend or pattern observed in the historical data.
- **Detecting anomalies**: Moving averages can be used to identify anomalies or outliers in the data by comparing the actual values to the moving average values.

Overall, moving averages are a simple but effective technique for analyzing time series data and can be used in combination with other techniques for more advanced analysis and prediction.

2.3.2 Rolling Window

The input and output sequence formats used for the problem under consideration in this thesis are created using the rolling window principle. A sequence of values is used as input while a single value is returned as output. The method used for data transformation is the sliding window or moving window. According to this principle, the data is divided into windows of a certain length, where each successive window sequence starts with the value one step later. For this experiment, the window length was set to 2 hours or 12 timestamps (equivalent to 10-minute averaged values). The timestamp represents a digital record of the time of an event. The sliding step is 1. The sliding window principle is shown in Figure 2.7.

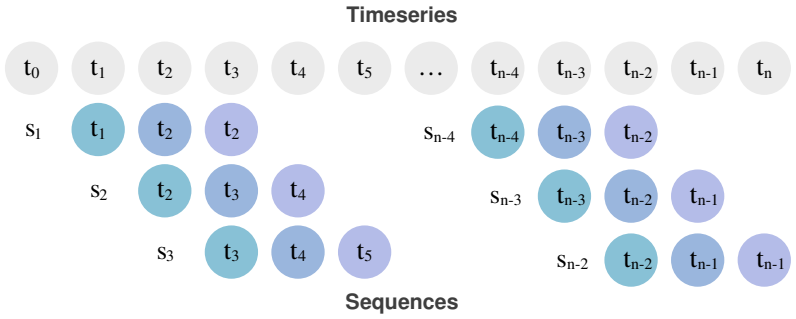


Figure 2.7: Rolling window method for a timeseries of n timestamps in which $n-(L-1)$ sequences of lenght $L = 2$ are made.

2.4 Machine Learning

Machine learning (ML) is an area of *Artificial Intelligence* (AI) concerned with the creation of algorithms that can learn from data and make predictions or judgments based on that data. The purpose of machine learning is to develop intelligent systems capable of improving their performance over time without being explicitly programmed.
Algorithms that employ machine learning learn from examples or data and use statistical methods to detect patterns and relationships in the data. Once a model has been trained, it can be applied to new data to generate predictions or judgments.
In contrast, AI is a broad field that incorporates a variety of methodologies and approaches for developing intelligent systems. Machine learning is one of the numerous subfields of artificial intelligence, alongside natural language processing, computer vision, and robotics.

2.4.1 Learning Paradigms

ML is frequently combined with other AI approaches to construct intelligent systems capable of complicated tasks such as speech recognition, natural language comprehension, and real-time decision making. Hence, machine learning is a crucial aspect of artificial intelligence, and the two sciences are tightly related. Machine learning can be categorized into three main distinct groups based on the learning strategy [19]:

- *Supervised learning* (SL) is a type of ML in which the model learns to make predictions based on labeled samples. Labeled data is data that has already been categorized or labeled with the appropriate output. In supervised learning, an algorithm is taught using a collection of inputs and outputs. The objective is to discover a mapping function capable of predicting the outcome of fresh, unobserved inputs. Regression and classification difficulties are examples of supervised learning.
- *Unsupervised learning* (UL) is a type of ML where the model discovers patterns in unlabeled data. The term "unlabeled data" refers to information that lacks a preset output or classification. In unsupervised learning, the algorithm is trained on a collection of inputs for which there are no outputs. The objective is to discover the underlying data structure and group comparable data points together. Unsupervised learning includes clustering and dimensionality reduction as examples.
- *Reinforcement learning* (RL) is a sort of ML in which a model learns by interacting with its environment and gets rewards or penalties as feedback. Its objective is to discover a strategy or collection of activities that maximizes the cumulative reward over time. Robotics and game play are examples of reinforcement learning.

There are also hybrid approaches, such as *semi-supervised learning*, which combines labeled and unlabeled data, and *self-supervised learning*, which uses unsupervised learning to pretrain a model before fine-tuning it on a supervised learning task.

Regression and Classification

Based on the desired output, supervised algorithms can be divided into:

- **Classification**: Is a process of categorizing or arranging things into various classes or categories based on their qualities or traits. In machine learning and data science, classification refers to the process of creating models that can learn to assign incoming data items to one of several predefined categories depending on their characteristics.
  The objective of classification issues is to create a predictive model that can learn from labeled data and accurately categorize fresh, unseen data points into one of the preset categories. Several applications, including image recognition, spam filtering, fraud detection, and sentiment analysis, make extensive use of classification.
- **Regression**: Is a statistical technique employed to determine the relationship between a dependent variable and one or more independent variables. The fundamental objective of regression analysis is to evaluate the connection between the dependent variable and the independent variables and make predictions about the first based on the values of the second. Regression analysis is utilized in numerous disciplines, including economics, finance, social sciences and engineering. In machine learning, regression is a supervised learning technique used to predict the continuous numeric values of a dependent variable based on one or more independent factors.

Clustering, Dimensionality Reduction and Anomaly Detection

There are numerous unsupervised learning strategies, including:

- **Clustering**: Clustering is a process that involves grouping together comparable data items based on their resemblance. The objective is to discover patterns and structures in the data without preconceived labels or classifications. Some clustering algorithms examples [20] are k-means, hierarchical clustering, and DBSCAN.
- **Dimensionality reduction**: Dimensionality reduction is a technique that reduces the number of variables or features in a dataset while maintaining the underlying structure of the data. The objective is to eliminate superfluous or irrelevant characteristics and simplify the data representation. Dimensionality reduction strategies include techniques such as *principal component analysis (PCA)* [21] , *t-SNE* [22], and *autoencoders*.
- **Anomaly detection**: Anomaly detection is a technique that identifies data points in a dataset that are exceptional or rare. The objective is to recognize patterns that depart from the norm and may signal unexpected behavior or occurrences. Algorithms for detecting anomalies are *isolation forest* [23], *local outlier factor (LOF)*, and *autoencoders*.

Types of Anomaly Detection

Anomaly detection can be categorized into three main types: supervised, unsupervised, and semi-supervised.

- **Supervised anomaly detection**: In this type, the algorithm is trained on labeled data, where each data point is labeled as either normal or anomalous. The algorithm then uses this labeled data to classify new data points as either normal or anomalous. Supervised anomaly detection is useful when there is a large amount of labeled data available for training.
- **Unsupervised anomaly detection**: In this type, the algorithm is trained on unlabeled data, where the data points are not labeled as normal or anomalous. The algorithm then tries to identify patterns in the data that are different from the majority of the data points, which are considered anomalies. Unsupervised anomaly detection is useful when there is no labeled data available for training.

- **Semi-supervised anomaly detection**: In this type, the algorithm is trained on both labeled and unlabeled data. The labeled data is used to train a classifier to identify normal and anomalous data points, while the unlabeled data is used to learn the underlying patterns in the data. Semi-supervised anomaly detection is useful when there is a limited amount of labeled data available for training. It's worth noting that unsupervised anomaly detection is more challenging than supervised anomaly detection because there is no labeled data to guide the learning process. The algorithm must identify anomalies solely based on the patterns and characteristics it learns from the unlabeled data. Therefore, it's important to carefully choose the features and techniques used for unsupervised anomaly detection to ensure that the algorithm can effectively identify anomalies in the data.

Each method of anomaly detection has benefits and drawbacks, and the choice of which type to employ relies on the nature of the problem and the data provided. When labeled data is available, for instance, supervised anomaly detection is more accurate than unsupervised anomaly detection. Nevertheless, it may be less effective at detecting novel forms of abnormalities that were not present in the labeled data. Unsupervised anomaly detection, on the other hand, can be more effective at detecting novel anomalies, but it may yield more false positives and false negatives.

In our test case, unsupervised anomaly detection is the appropriate approach. If a large number of anomalies and strange behavior is provided and there is no way in how to label them, then supervised anomaly detection is not feasible.

Classification and Anomaly Detection: Similarities and Differences

Binary classification and *anomaly detection* are two machine learning techniques used for different purposes, although they may seem similar at first glance. Both techniques involve making a decision based on input data, but they differ in terms of the goal of the analysis, the nature of the data, and the techniques used to analyze it.

Binary classification is used to classify data into one of two classes or categories. For example, in medical diagnosis, binary classification can be used to determine if a patient has a certain disease or not. In this case, the two classes are "disease" and "no disease". Binary classification involves training a model on a labeled dataset in order to predict the class of new, unseen data. The goal of binary classification is to minimize the number of misclassifications, which are the cases where the model assigns the wrong class.

Anomaly detection, on the other hand, is used to identify rare events or observations that deviate significantly from the expected behavior of a system. In contrast to binary classification, anomaly detection is often an unsupervised learning technique, where the data is not labeled as normal or anomalous. it can be used in various domains, such as cybersecurity, fraud detection, and predictive maintenance. Its goal is to detect the unusual or rare events, rather than classify data into specific categories.

One of the key differences between binary classification and anomaly detection is the nature of the data. In binary classification, the data is often balanced, meaning that the number of instances in each class is roughly equal. In anomaly detection, the data is typically imbalanced, where the majority of observations are normal and only a small proportion are anomalous. This means that standard classification techniques may not work well for anomaly detection, and specialized techniques such as outlier detection, clustering, or density-based methods may be needed.

Another key difference is in the evaluation of the models. In binary classification, the models are typically evaluated using metrics such as *accuracy*, *precision*, *recall*, and *F1 score*. In anomaly detection, the evaluation is often more challenging because the dataset is imbalanced, and the true labels are not known. Therefore, evaluation metrics such as the area under the *receiver operating characteristic curve (AUC-ROC)* or the area under the *precision-recall curve (AUC-PR)* are used to measure the performance of the model.

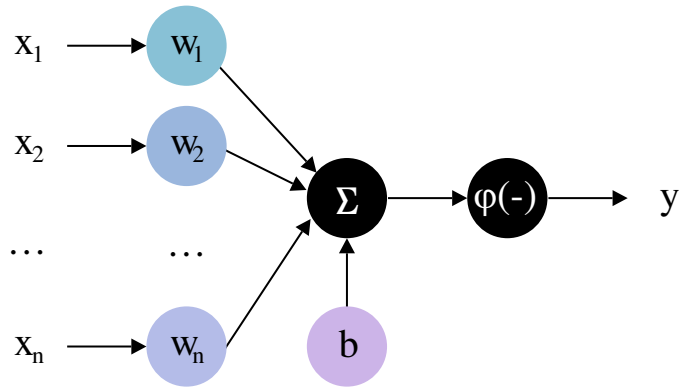


Figure 2.8: Schema representing the artificial neuron with inputs and outputs. In the process, an input x_n is multiplied with a set of weight w_n with summed with the bias b, all aggregated by a function Σ. The output from the aggregate function is the input of the activation function ϕ with output y.

2.4.2 Artificial Neural Network

Neural networks are a machine learning technique that are based on the structure and function of biological neurons. It is made up of interconnected artificial neurons that transmit and process information via a network of weighted connections. Neural networks are employed for a variety of tasks, such as pattern recognition, classification, regression, and prediction.
The fundamental component of a neural network is an artificial neuron, which is a mathematical function that accepts one or more inputs, computes a weighted sum of those inputs, and applies an activation function to generate an output. Training is essential to determine ites weights and activation function.

The concept of neural networks dates back to the 1940s, but it wasn't until the 1980s and 1990s that significant advancements were made in creating effective training algorithms and architectures for neural networks. Since then, neural networks have been utilized in numerous applications, such as image recognition [24], speech recognition [25], natural language processing [26], and autonomous vehicles [27]. There are numerous types of neural network models, each with its own architecture and training algorithms that make them suitable for a variety of tasks. Examples of common neural network types include *Feed-Forward Neural Network* (FNN), *recurrent neural networks* and *convolutional neural networks*.

Adjusting the weights of their connections based on the error signal generated by the cost function is how neural networks learn. This method is commonly referred to as supervised learning because it requires a labeled dataset to train the network. Autoencoders and generative adversarial networks are two examples of unsupervised learning techniques that can be used to train neural networks.
The number of layers, the number of neurons in each layer, the learning rate, and the activation function are among the hyperparameters that must be set prior to training neural networks. Selecting the appropriate hyperparameters is crucial for achieving good performance and avoiding overfitting and underfitting [28].

Using a technique known as backpropagation, neural networks are trained by iteratively adjusting the weights of the connections between neurons based on the difference between the network's predicted and actual outputs for a given set of training data [29]. The objective of training is to minimize a cost function that measures the gap between predicted and actual outputs.
The gradients are then employed to update the weights in the opposite direction of the gradient, thereby decreasing the cost function. The learning rate is a hyperparameter that controls how frequently the weights are updated during each iteration of backpropagation. The cost function is a function that measures the gap between the predicted and actual outputs for a given set of training data.

In Figure 2.8 the structure of an artificial neuron is shown alongside the description of its logic. Each of the n inputs, represented by x_1, x_2, x_n, is multiplied by its relative weight and the

result is added with a value called bias to obtain the net input function Σ. The bias allows for the introduction of a shift into the activation function [30]. The activation function determines the calculation of the output. The result proceeds to the activation function ϕ, which determines the calculation of the output and if a neuron should be activated or not. The scope is to decide whether or not the input of the neuron provided to the network is significant during the prediction process.

Artificial Neural Network is composed of multiple neurons called nodes connected to each other through links called synapses. A simple ANN consists of at least three layers of simple neurons. The first layer, called the input layer, is followed by the hidden layer in the middle and an output layer. The neurons in the layers are interconnected and a weight is assigned to each connection.

During the training phase, the output values are compared with labels and the result is called the loss function or cost function. The loss function is used in the back-propagation process to adjust the output results, which means that the error in the output layer is carried back through the neural network. Instead, during the testing phase, a new set of data is sent to the neural network and the output error value represents the value of the metric. ANNs are part of supervised learning as they normally require labeled data to learn the task to do.

Perceptron: The Foundation of ANN

The concept of ANN was first proposed by Warren McCulloch and Walter Pitts in 1943 and was then implemented by Frank Rosenblatt in 1958 as *perceptron*.
A *perceptron* is a type of artificial neural network that can be used for supervised binary classification tasks. It consists of a single layer of neurons, where each neuron computes a weighted sum of its inputs and applies a nonlinear activation function to the result. The output of the *perceptron* is then passed through a threshold function to generate the final binary classification decision.

Let x = (x1, x2, ..., xn) be the input vector of n features, and w = (w1, w2, ..., wn) be the weight vector. The weighted sum of inputs is computed as:

$$z = \sum_{i=1}^{n} w_i x_i$$

Then, a nonlinear activation function f is applied to z to obtain the output y:

$$y = f(z)$$

Common activation functions used in *perceptrons* include the step function, *sigmoid* function, and $ReLU$ function. Finally, the output y is passed through a threshold function g to generate the binary classification decision:

$$\hat{y} = g(y)$$

The threshold function is typically defined as:

$$\hat{y} = \begin{cases} 1, & \text{if } y \geq \theta \\ 0, & \text{otherwise} \end{cases}$$

where θ is the decision boundary or threshold value.

The *perceptron* learning algorithm involves updating the weights based on the error between the predicted output and the true label. Let d be the true binary label of the input x. The error e is defined as:

$$e = d - \hat{y}$$

The weights are then updated according to the following rule:

$$w_i \leftarrow w_i + \eta e x_i$$

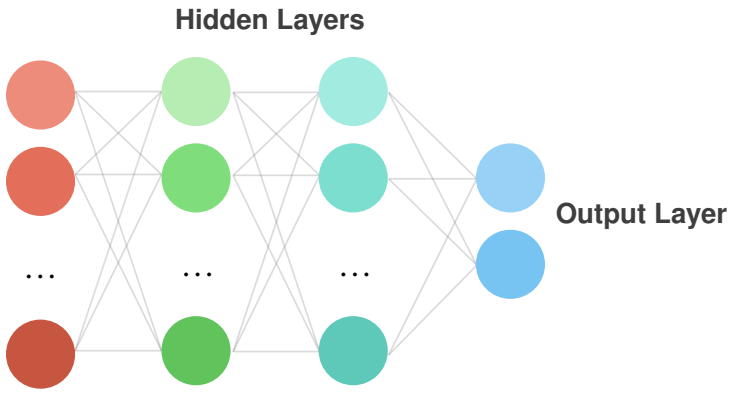


Figure 2.9: Generic Multilayer Perceptron configuration.

where η is the *learning rate*, which controls the *step size* of the *weight updates*. In summary, the *perceptron* is a simple but powerful algorithm for binary classification tasks. It involves computing a weighted sum of inputs, applying a nonlinear activation function, and passing the output through a threshold function to generate the final binary classification decision. The weights are updated based on the error between the predicted output and the true label using a simple rule.

Deep Learning and Multilayer Perceptron

Deep learning is a subfield of machine learning that involves training neural networks with multiple layers to learn and make predictions from large datasets. A neural network is a collection of connected nodes (neurons) that are interconnected and structured in layers. The layers closest to the input data are called input layers, and the layers closest to the output predictions are called output layers. The layers in between are called hidden layers [31].

The *Multilayer Perceptron* (MLP) is a type of neural network that consists of multiple layers of perceptrons (single-layer neural networks) (as shown in Figure 2.9). The MLP is one of the simplest and most popular types of FNN. In the hidden layers of an MLP, the perceptrons receive weighted inputs from the previous layer, apply a non-linear activation function to the sum, and send the result to the next layer. MLPs are often used for supervised learning tasks, such as classification and regression, and may be taught via backpropagation.

MLPs are a form of deep learning since they comprise numerous layers of neurons that are linked. Nevertheless, they are not as deep as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), which are used in deep learning (RNNs). Nonetheless, MLPs can still be very effective for certain tasks and are relatively easy to implement and train.

2.4.3 Recurrent Neural Network

Recurrent Neural Networks (*RNNs*) are ANNs that, differently from FNNs which only admit connections between nodes in different layers, allow connections even within the same layer. This characteristic is depicted in Figure 2.10. In RNNs, nodes can also admit loops and/or can also be connected with neurons from a previous layer. This feature allows RNNs to "remember" the previous input while processing a new one. In an RNN network, the output of a neuron can influence both itself and other neurons at a later time, which in turn will again interfere with the behavior of the neuron by forming a loop.

An RNN network can be represented by a cell with a loop. Through a network unfolding operation, depicted in Figure 2.10, the RNN is transformed into a feed-forward one.

An RNN cell is a recurrent network section that preserves an internal state *h(t)* for each time instant. A cell consists of a fixed number of neurons and can be considered a kind of layer. In this network at each instant the output will be $h_t = f(h_{(t-1)}, X_t)$ where h_t depends on the input X_t and the previous state $h_{(t-1)}$.

The equations of the RNN states, during the learning process are:

- Hidden state update:
$$h_t = f(W_{xh}x_t + W_{hh}h_{t-1} + b_h) \tag{2.1}$$
- Output computation:
$$y_t = g(W_{hy}h_t + b_y) \tag{2.2}$$

In these equations, x_t is the input at time step t, h_{t-1} is the previous hidden state, W and b are weight and bias parameters, f and g are activation functions (such as sigmoid, tanh, or ReLU) [32], and y_t is the output at time step t. During training, the parameters W and b are updated through backpropagation through time (BPTT) to minimize a loss function, such as the mean squared error or cross-entropy loss. BPTT is a variant of backpropagation that computes the gradients of the loss with respect to the parameters at each time step, and accumulates them over the entire sequence. The learning process in an RNN can be prone to the vanishing gradient problem, where the gradients can become very small as they are backpropagated through time, making it difficult to learn long-term dependencies. To mitigate this problem, variants of RNNs have been proposed, such as LSTMs and GRUs, which use gating mechanisms to selectively update the hidden state and control the flow of information through the cell.

The input and output of RNN models can be single data or sequences in all combinations (showed in Figure 2.11). Particularly for regression problems, these versions are implemented:

- Many-to-one with a sequence as input and a single time as output.
- Many-to-many with both input and output sequences.

Time series with many inputs suffer from the problem of disappearing or exploding gradients, and this happens because updating the weights for an output requires numerous multiplications that cause the weights to tend to zero or infinity [33].

Long Short-Term Memory

Long Short-Term Memory (*LSTM*) neural networks are particular RNNs capable of learning dependencies in long time series data, and they solve the problem of the vanishing of gradient Like all RNNs, LSTMs can be considered as a set of memory cells. The LSTM architecture was firstly introduced in [34], and the cell main feauture is that it is able to control how much information to remember from the previous cell, how much information to retain from the current cell, and how much to inject into the next cell. These tasks are handled by the gates of the cell. (Figure: 2.12). Each LSTM hidden layer has as many hidden cells as the number of time steps. Moreover, each hidden cell is composed of multiple hidden units. The unit represents the number of neurons per cell.
The learning process in an LSTM cell involves updating the values of the cell state and hidden

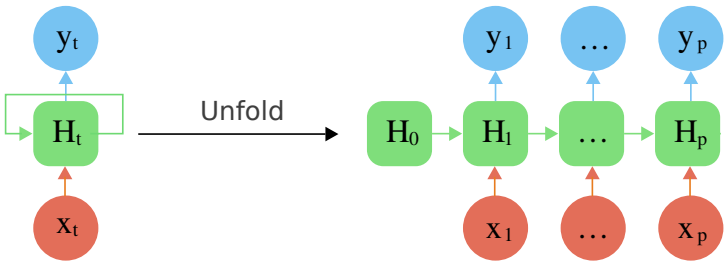


Figure 2.10: Unfolding operation performed on a RNN cell.

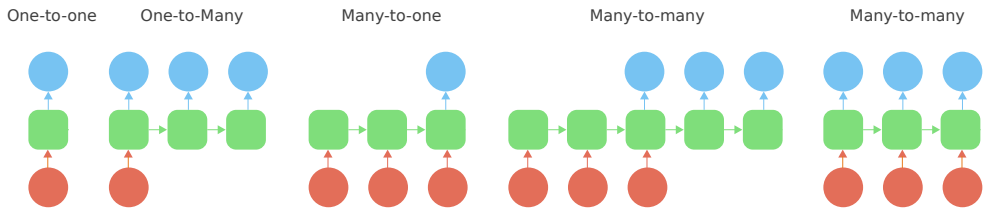


Figure 2.11: Different RNN working modes (and examples), from left to right: One-to-one (no sequential), one-to-many (text-to-X generation), many-to-one (sentiment analysis), many-to-many with different input and output length (machine translation) and many-to-many (data reconstruction). The red rectangle represent the inputs, while the blue ones the algorithm outputs. The green rectangle holds the intermediate state.

state based on the input, previous hidden state, and previous cell state. The input gate determines which information from the input should be added to the cell state, while the forget gate determines which information from the previous cell state should be discarded. The output gate determines which information from the cell state should be output as the new hidden state.
The first is the input gate, which determines the information to add to the cell state:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \tag{2.3}$$

Next, is the forget gate, which determines the information to discard from the cell state:

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \tag{2.4}$$

Then, there is the cell state update, which determines the new cell state value:

$$c_t = f_t c_{t-1} + i_t \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \tag{2.5}$$

Finally, the output gate, which determines the information to output as the new hidden state:

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \tag{2.6}$$

$$h_t = o_t \tanh(c_t) \tag{2.7}$$

In these equations, x_t is the input at time step t, h_{t-1} is the previous hidden state, c_{t-1} is the previous cell state, W and b are weight and bias parameters, and σ is the sigmoid function. The tanh function is used to squash the cell state values between -1 and 1, which helps prevent the vanishing gradient problem.
During training, the parameters W and b are updated through backpropagation to minimize a loss function, such as the mean squared error or cross-entropy loss. The goal is to learn a set of weights that can accurately predict the target output given the input sequence.

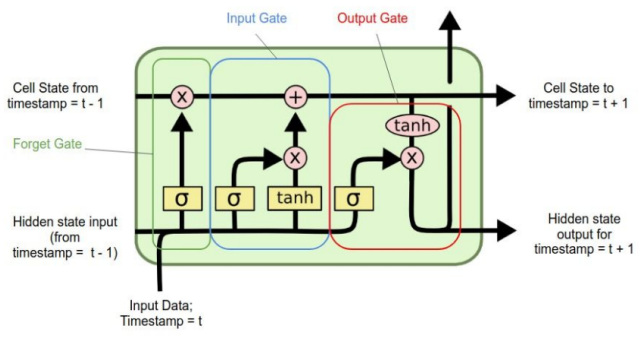


Figure 2.12: LSTM Cell. Image from [35].

Gated Recurrent Unit

Firstly introduce in [36], Gated recurrent unit (GRU), is a simplified version of LSTM that was developed to solve some of its architectural restrictions. Two gates, a reset gate and an update gate, regulate the information flow inside a GRU (Figure 2.13). The reset gate controls how much information from the past should be discarded, while the update gate defines how much information from the present should be added.
The primary distinction between GRU and LSTM is their architectural complexity and gate count. LSTM contains three gates, a memory cell, and a large number of parameters to learn, whereas GRU has just two gates and a smaller number of parameters. GRUs are hence easier to train and require less data to generalize effectively, but they may not perform as well as LSTMs on memory-intensive complicated tasks.
In fact, both GRU and LSTM are useful for modeling sequential data, and the selection between the two relies on the specific job and amount of training data.
Just like the LSTM the learning process, involves updating the cell state and hidden state based on the input. The gates are updated with the following equations:

- Reset gate:
$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r) \tag{2.8}$$
- Update gate:
$$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z) \tag{2.9}$$
- Candidate hidden state:
$$\tilde{h}_t = \tanh(Wxhx_t + r_t(W_{hh}h_{t-1}) + b_h) \tag{2.10}$$
- Hidden state update: Hidden state update:
$$h_t = (1 - z_t)h_{t-1} + z_t\tilde{h}_t \tag{2.11}$$

In these equations, x_t is the input at time step t, h_{t-1} is the previous hidden state, W and b are weight and bias parameters, σ is the sigmoid function, and $\tilde{h}_t$ is the candidate hidden state. The reset gate r_t and the update gate z_t are used to control the flow of information in the cell.
During training, the parameters W and b are updated through backpropagation to minimize a loss function, such as the mean squared error or cross-entropy loss. The goal is to learn a set of weights that can accurately predict the target output given the input sequence.

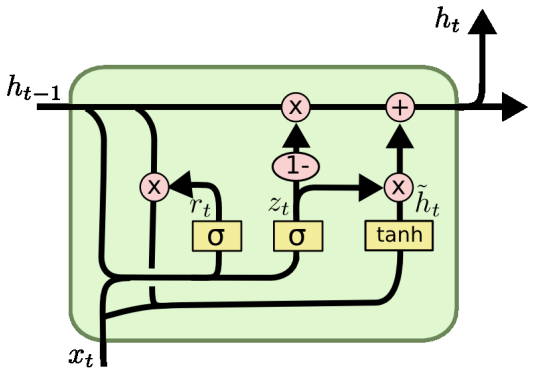


Figure 2.13: GRU Cell. Image from [37]

2.4.4 Autoencoders

An *autoencoder* (AE) is a type of neural network that can are trained to reconstruct their input data by first learning a compact representation of the input in a lower-dimensional space, known as the encoding, and then using this encoding to recreate the original data. An AE architecture (Figure 2.14) consists of two sub-components: the **encoder** and the **decoder**.

The *encoder* converts an input to a lower-dimensional representation, typically referred to as a *bottleneck* or *latent space*. Typically, the *encoder* comprises of one or more neural network layers that perform *feature extraction* and *dimensionality reduction*. The lower-dimensional space, which is the intermediate representation of the input data that an AE learns to extract, may then be utilized for activities such as visualization, grouping, and further analysis, making it simpler to comprehend the relationships and patterns in the original data.
Typically, the size of the latent space is substantially lower than the amount of the input data, allowing the AE to successfully capture the key aspects of the input while disregarding noise. The choice of latent space size is a crucial hyperparameters in the AE, as it influences the tradeoff between the quality of the reconstructions and the quantity of information lost during the encoding process. In general, a smaller latent space will provide simpler reconstructions with less information, whereas a larger latent space will produce more comprehensive reconstructions with a greater danger of overfitting [38].

In symbols, let $\mathbf{x} \in \mathbb{R}^n$ be an input vector, and let $\mathbf{z} \in \mathbb{R}^m$ be the corresponding latent code, where $m < n$. The *encoder* maps the input $\mathbf{x}$ to the latent code $\mathbf{z}$ using a function $f_\theta(\mathbf{x})$, where θ are the learnable parameters of the *encoder*. Formally:

$$\mathbf{z} = f_\theta(\mathbf{x})$$

The *decoder* translates the representation of the bottleneck back to the original input space. Typically, the *decoder* is comprised of one or more neural network layers that conduct feature expansion and reconstruction. The output layer is the last layer of the *decoder*, and it has the same amount of neurons as the input layer. The *decoder* maps the latent code $\mathbf{z}$ back to the original input space using a function $g_\phi(\mathbf{z})$, where ϕ are the learnable parameters of the *decoder*. Formally, where $\hat{\mathbf{x}}$ is the reconstructed input:

$$\hat{\mathbf{x}} = g_\phi(\mathbf{z})$$

The AE is trained by minimizing the reconstruction error between the input $\mathbf{x}$ and the reconstructed input $\hat{\mathbf{x}}$. This is typically done by minimizing the mean squared error (MSE) between the two:

$$L(\mathbf{x}, \hat{\mathbf{x}}) = ||\mathbf{x} - \hat{\mathbf{x}}||_2^2$$

The overall objective of the AE is to find the optimal values of the *encoder* and *decoder* parameters θ and ϕ that minimize the average reconstruction error over a set of training examples:

$$\min_{\theta,\phi} \frac{1}{N} \sum_{i=1}^{N} L(\mathbf{x}^{(i)}, \hat{\mathbf{x}}^{(i)})$$

where N is the number of training examples, and $\mathbf{x}^{(i)}$ and $\hat{\mathbf{x}}^{(i)}$ are the input and reconstructed input, respectively, for the ith training example.

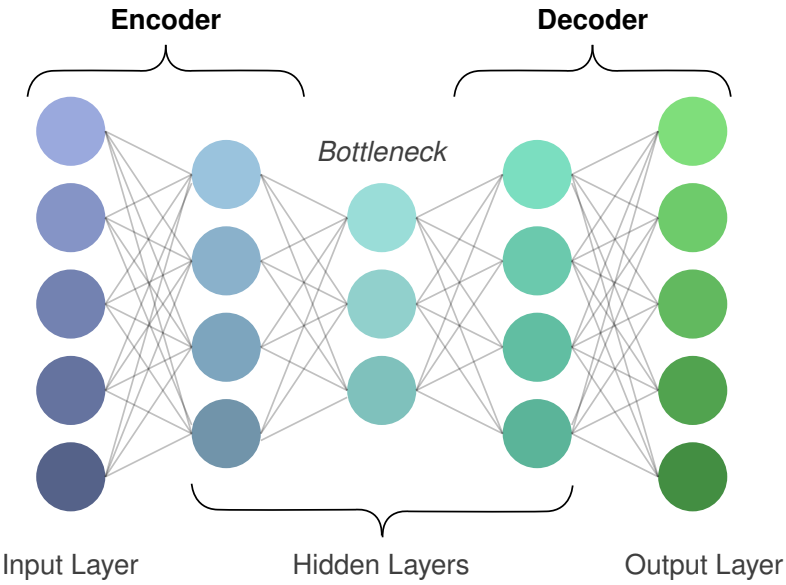


Figure 2.14: Simple AE architecture, with 3 hidden layers. The *encoder* network is in charge of transforming the 5 dimensional input in the corresponding 3 dimensional *latent space* representation, while the *decoder* recovers the original dimensionality from the *bottleneck* layer.

Autoencoder for time series reconstruction

In time series reconstruction, an AE can be taught a compact representation of the underlying patterns and dependencies in the data by being trained on a subset of the time series. This encoding can then be used to rebuild missing or invisible time series values.
A typical AE design for time series reconstruction consists of an *encoder* network that maps the input time series to a lower-dimensional representation and a *decoder* network that maps the encoding back to the original time series. The network is trained to *minimize* the difference between the *original* and *reconstructed* time series, which is defined as the *loss function*.
AE can be used to rebuild missing or corrupted data in numerous forms of time series data, including financial, ecological, and meteorological time series [39]. They have been demonstrated to improve the precision of time series reconstruction and can be utilized as a preprocessing step in other time series analytic jobs. AEs are ideally suited for time series analysis due to their ability to capture nonlinear relationships and patterns in the data.

AEs have been widely used in time series data analysis, and have been applied to a variety of tasks, including:

- **Anomaly detection**: AEs can be used to identify anomalies in time series data, such as unusual patterns or deviations from normal behavior [40].
- **Forecasting**: AEs can be used for time series forecasting, such as predicting future values of a given time series.
- **Dimensionality reduction**: AEs can be used to reduce the dimensionality of time series data, allowing for more efficient processing and storage.
- **Representation learning**: AEs can be used to learn compact representations of time series data, which can be used for various tasks, such as clustering, classification, and visualization.
- **Data denoising**: AEs can be used to remove noise (Figure 2.15 illustrates different data in a visual denoise task) from time series data, making it easier to identify patterns and trends in the data.

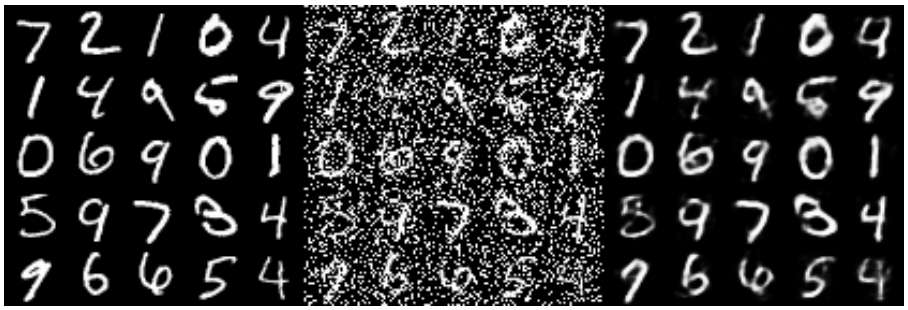


Figure 2.15: Different data in a denoising problem on MNIST dataset. From left to right: original input data, corrupted data, reconstructed data.

RNN Based Autoencoders

An AE based on *Recurrent Neural Networks* (RNNs) is a neural network architecture used for reconstruction of sequence-to-sequence data. It is a variation of the AE architecture used for unsupervised learning of *feature representations* from input data [41]. *Encoder* and *decoder* networks of an RNN-based autoencoder are both constructed of RNNs, which are used to encode and decode sequential data.

An RNN-based autoencoder's design consists of two major components: an *encoder* network and a *decoder* network. The encoder network receives a series of data as input and encodes it using an RNN into a lower-dimensional representation (*latent space*). The decoder network receives the encoded sequence as input and uses another RNN to recover the original sequence. During training, the reconstruction error, which quantifies the difference between the original sequence and the rebuilt sequence, is commonly employed as the loss function.
Formally, the equations for the RNN-based autoencoder training process are the following: Encoder network:

$$h_t = f_e(W_{xh}x_t + W_{hh}h_{t-1} + b_h) \tag{2.12}$$

$$z = h_T \tag{2.13}$$

Decoder network:

$$h_t = f_d(W_{zh}z + W_{hh}h_{t-1} + b_h) \tag{2.14}$$

$$\hat{x}t = g(Whxh_t + b_x) \tag{2.15}$$

In these equations, x_t is the input sequence at time step t, h_{t-1} is the previous hidden state, W and b are *weight* and *bias* parameters, f_e and f_d are activation functions used in the encoder and decoder RNNs respectively, z is the encoded sequence, $\hat{x}_t$ is the reconstructed output at time step t, and g is an activation function used in the output layer.
During training, the parameters W and b are updated using *backpropagation through time* (BPTT), which computes the *gradients* of the *reconstruction error* with respect to the parameters at each time step and accumulates them over the entire sequence.

AE based on RNNs have been implemented in a variety of applications, including speech recognition, machine translation, and anomaly detection in time series data. They may also be developed to more complicated designs, such as *variational autoencoders* [42] and *adversarial autoencoders* [43], which employ additional regularization and adversarial training approaches to increase the quality of the learnt representations.

RNN Returning Hidden States

When stacking several RNN layers, the decision between returning the complete sequence or simply the hidden state is driven by the goal to minimize the dimensionality of the input to the subsequent layer. By returning the whole sequence from each layer, all hidden states of the current layer are transmitted as input to the subsequent layer, resulting in a high-dimensional representation. Nonetheless, this method might be advantageous if the information included in

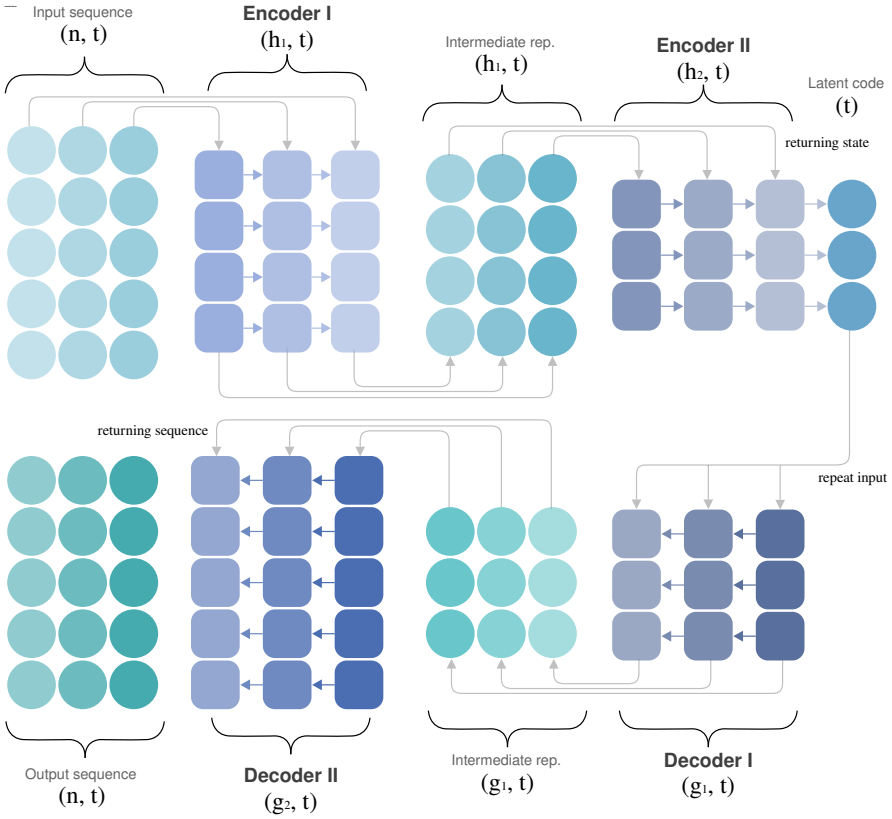


Figure 2.16: Example schema of the data flow in an RNN-based AE. Starting from an input sequence of n measures and t timesteps, data is processed through the different encoder layers. In the first encoder sequence are return, while in the second one, only the final state becomes the *latent code*. An RNN layer with h units produces output of h measures and t timesteps, transforming the first dimension and leaving unchanged the one representing time. The last layer of the AE, in this case, is a RNN layer with n units, instead of h_1 dimension. This choice has been done to reduce the image size but the last decoder layer can also be similar to the first encoder layer with number of units. In that other case the last layer will be a dense one, which has to map the decoder output to the effective newtork output.

the complete sequence is essential for the final prediction [41]. Returning just the last hidden state from each layer, on the other hand, minimizes the dimensionality of the input, since only the final hidden state is transmitted to the next layer (Figure 2.16). This strategy might be effective when the objective is to capture the overall context of a sequence rather than granular information at each time step. The trade-off is that some information from the sequence is lost, although this might be a suitable compromise when the dimensionality of the input is an issue.

Variational Autoencoders

A *variational autoencoder* (VAE) is a type of generative neural network model [44] that learns to produce new data by encoding input data into a low-dimensional latent space and decoding the latent representation back into the original data space.
In a VAE, the encoder converts an input data point x to a typical *Gaussian probability distribution* in the latent space. A point in the latent space is then mapped by the decoder to a probability distribution over the original data space, which is commonly a Gaussian distribution. The network is trained by optimizing the evidence lower limit, which is a lower bound on the log-likelihood of the training data (ELBO).
The ELBO may be broken down into two components: *reconstruction loss* and the *KL divergence* between the learnt latent distribution and a previous distribution. The *reconstruction loss* is a measure of the network's ability to reconstruct the input data from the *latent representation*,

while the *KL divergence* term encourages the network to learn a latent distribution that is close to the prior distribution [45].
In symbols, ELBO can be written as:

$$\text{ELBO} = \mathbb{E}_{q(z|x)}[\log p(x|z)] - \text{KL}(q(z|x)||p(z))$$

where $p(x|z)$ is the *decoder probability distribution*, $q(z|x)$ is the *encoder probability distribution*, and $p(z)$ is the *prior distribution* over the latent space.

The objective of a VAE is to discover a compact and useful representation of the input data, which can then be used to produce new data samples that are comparable to the training data. VAEs are often utilized in applications such as the production of images, videos, and text.
Additional loss terms can be added to the VAE objective function to improve its performance for certain jobs. For instance, the addition of a reconstruction loss term might encourage the VAE to create more realistic samples. Moreover, a *regularization term* may be introduced to the encoder to promote sparsity in the learnt representation. These extra loss terms can aid in enhancing the VAE's performance for specific tasks.
The VAE objective function is defined as follows:

$$\mathcal{L}\text{VAE} = -\mathbb{E}z \sim q(z|x)[\log p(\tilde{x}|z)] + \text{KL}(q(z|x)||p(z))$$

where $\text{KL}(q(z|x)||p(z))$ is the *Kullback-Leibler* divergence between the encoder distribution and a prior distribution $p(z)$, which is typically a standard *normal distribution*. The $-\mathbb{E}_{z\sim q(z|x)}[\log p(\tilde{x}|z)]$ represents the reconstruction error, while the term $\text{KL}(q(z|x)||p(z))$ is the KL divergence regularization term.

2.4.5 Overfitting

Overfitting is a machine learning issue in in which the output of an analysis corresponds too closely to a certain data set. It occurs when a model is *over-trained* on the training data, to the point that it begins to *fit* the *noise* and random changes rather than the *underlying patterns* (as shown in Figure 2.17). Consequently, the model may perform well with training data but badly with validation or test data. This is due to the fact that the model has learnt to memorize the training data, as opposed to generalizing to new, unknown data.
Overfitting in machine learning can be explained by analyzing the model's representation of the *prediction-making* space. A machine learning model will typically discover a mapping between input data and output predictions. The space in which this mapping is learned is called the *hypothesis space*. This indicates that the model fits the training data too well, leading in a hypothesis space that is too particular to the training data and does not transfer well to new, unobserved data.

Overfitting can be caused by a number of factors in machine learning [46], including:

- **Complex model architecture**: A model with too many parameters relative to the size of the training data can easily overfit, as it has the capacity to fit the noise in the training data.
- **Insufficient training data**: A model trained on a small training dataset is more likely to overfit, as it may not have seen enough diverse examples to generalize to new, unseen data [47].
- **High model capacity**: A model with high capacity, such as deep neural networks, can overfit more easily, as it has the ability to fit very complex and intricate patterns in the data.
- **Unregularized training**: Training a model without regularization can result in *overfitting*, as the model will try to fit the training data as closely as possible.
- **Overly long training**: Training a model for too long can cause *overfitting*, as the model may continue to fit the training data even when it has already learned the underlying patterns [48].

Using techniques like as *regularization* as well as a bigger and more varied training dataset helps avoid or limit *overfitting*. During training, it is essential to check the model's performance on a validation set to detect this issue and make any required modifications.

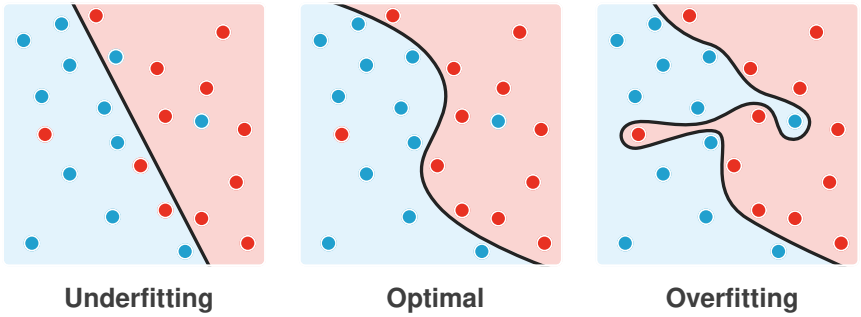


Figure 2.17: Underfitting and overfitting representation schema in a classification process.

2.4.6 Regulatization

Regularization is a method used to avoid *overfitting* in neural networks by adding a penalty term to the loss function. When a model is excessively complicated and learns the noise in the training data rather than the underlying patterns, *overfitting* occurs. Regularization aims to enhance the generalization performance of a model by lowering its ability to learn irrelevant input characteristics or noise.

Regularization for RNN

The following are examples of common RNN regularization techniques:

- **Dropout** is a strategy in which a random subset of neurons are momentarily *"dropped out"* or *turned off* in each forward pass during training. By requiring the network to learn numerous distinct representations of the input data, this helps prevent *overfitting* [49].
- **L1 and L2 regularizations** add a penalty term to the loss function to encourage the network to have smaller weights. L1 regularization produces sparse weight matrices, whereas L2 regularization favours small and evenly distributed weights [50].
- **Early stopping** entails monitoring the network's performance on a validation set and terminating the training process when the performance on the validation set begins to decrease.
- **Weight decay** is a method of regularization that promotes the network to have smaller weights by adding a penalty term to the loss function that is proportional to the magnitude of the weights.
- **Data Augmentation** increases the quantity of the training data by generating additional instances by transformations like as rotation, scaling, and inversion. The most effective regularization strategy relies on the RNN's architecture and workload, and frequently requires testing to find. Regularization is an essential part of training RNNs and can significantly enhance the network's generalization performance on unknown input.

Denoise Autoencoders

Denoising autoencoder (DAE) is a form of AE that is trained to reconstruct the original, uncorrupted input data from a corrupted version of the input [51]. The goal of the DAE is to discover a robust representation of the input data that can successfully reduce noise.

In order to reproduce this process, the first step is to generate a version of the input data with noise: Noise is added to the input data by masking numbers at random, introducing Gaussian noise or otherwise polluting the data.

In symbols, let $\mathbf{x} \in \mathbb{R}^n$ be a noisy input vector, and let $\mathbf{z} \in \mathbb{R}^m$ be the corresponding latent code, where $m < n$. To add noise to the input, a noise function $h(\cdot)$ that corrupts the input in a controlled way, is introduced. For example, by adding a Gaussian noise with zero mean and variance σ^2 to each element of the input:

$$\tilde{\mathbf{x}} = h(\mathbf{x}) = \mathbf{x} + \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(0, \sigma^2 \mathbf{I})$$

The objective function of the problem, as well as the mapping function provided by the *encoder* and *decoder*, are unchanged respect to the previous case. During train the AE learns to recover the uncorrupted, original input data from the noisy version: which corresponds to reduce reconstruction loss between the original input data and the reconstructed output.

DAE can be seen of data augmentation strategy. During training, the model is exposed to many permutations of the input data and learns to recognize the underlying patterns while ignoring the noise. By training on noisy data, the model is compelled to acquire a more robust representation of the input data, which might enhance its generalization performance. This is comparable to the concept of *data augmentation*, in which the training data is increased by the application of numerous modifications to the original data [52].

Dropout

Using dropout in a neural network can help prevent overfitting and improve generalization performance by reducing the co-adaptation of neurons, which can lead to over-reliance on specific features. By randomly dropping out neurons during training (Figure 2.18), the network is forced to learn more robust and generalizable features, which can improve its performance on unseen data. Let w be the weights of a network layer with biases and x be the input to the layer. The output of the layer is given by:

$$y = f(w \cdot x + b)$$

where f is the activation function of the layer and b is the bias term.

To apply dropout to the layer, a dropout mask m is introduced, which is a binary vector of the same size as the output y. The mask is generated by sampling from a Bernoulli distribution with a parameter p, which represents the probability of dropping out each neuron. The mask is then multiplied element-wise with the output y to obtain the masked output $\tilde{y}$:

$$m \sim Bernoulli(p)$$

$$\tilde{y} = m \odot y$$

where $\odot$ denotes element-wise multiplication.

During training, the dropout mask is sampled anew for each forward pass through the layer to create a different mask for each mini-batch of data. During testing or inference, the mask is not used, and the full output y is used instead.

Dropout and Overfitting

Dropout may be used as a kind of regularization to minimize overfitting in anomaly or outlier identification [53]. When training an anomaly detection model, the objective is to identify occurrences that differ from the regular behavior pattern as anomalies. Overfitting may result in a model that is overly particular to the training data and may not generalize to new data well. In this scenario, dropout might be advantageous by introducing unpredictability into the model, which reduces the likelihood of the model overfitting the training data. A model that is well-regularized is more likely to generalize and discover abnormalities in fresh, unobserved data. On

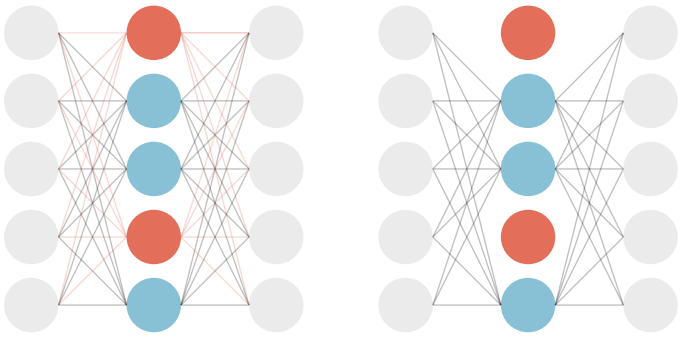


Figure 2.18: Dropout schema in which 2 neurons (in red) are deactivated. In left image the weight related to the deactivated neuron are in red while on the right the weights are removed.

the other side, overfitting the model to the training data may result in a model that is too particular to the training data and does not generalize well to new data, leading to a large number of false positive or false negative predictions. Overfitting may not be advantageous for anomaly detection since it may result in a model that is not generalizable and ineffective at finding abnormalities in fresh data. In conclusion, it is preferable to utilize dropout or other types of regularization in anomaly or outlier detection models, as it prevents overfitting and increases the generalizability of the model.

Dropout and DAE

DAE and Dropout in an AE are two regularization strategies in deep learning that are distinct from one another.
DAE are taught to rebuild a clean input from a noisy version of the same input. During training, the input is contaminated randomly with zero values (or other types of noise), and the model is taught to recreate the original, uncorrupted input. The objective is to get a robust representation of the input that can withstand a certain amount of noise. Dropout, on the other hand, is a regularization strategy that randomly sets to zero a portion of the network's activations during training. This reduces *overfitting* by preventing the network from depending too much on a single characteristic. Dropout may be implemented on any network layer, not simply auto-encoders. DAEs are a form of auto-encoder trained with a particular type of noise, zero values, to acquire a robust representation, while dropout is a universal regularization approach that can be used to any layer of a deep learning network to minimize *overfitting* [51]. DAE may be achieved using input dropout in the first layer. In this instance, the dropout layer would function as a sort of noise injection by randomly setting a portion of input activations to zero during training [54]. It is essential to note, however, that dropout is frequently performed at a substantially greater rate (e.g., 50%) than denoising in DAEs (normally 10%). Typically, the dropout rate for DAEs is significantly lower to guarantee that sufficient information remains in the input for the model to develop a usable representation [55].
In conclusion, dropout may be used as a sort of noise injection in a DAE to avoid *overfitting* and regularize the model, however the rate of dropout employed for DAEs is often lower than for other regularization strategies such as dropout in a FNN.

2.4.7 Principal Component Analysis

Principal Component Analysis (PCA) is a method that is commonly utilized in the field of multivariate data analysis. Its purpose is to reduce the dimensionality of a dataset while maintaining the majority of that dataset's variability. The fundamental objective of principle component analysis (PCA) is to convert an initial collection of correlated variables into a new set of uncorrelated variables known as principal components. These principal components are linear combinations of the variables that were used in the analysis (schema in Figure 2.19).

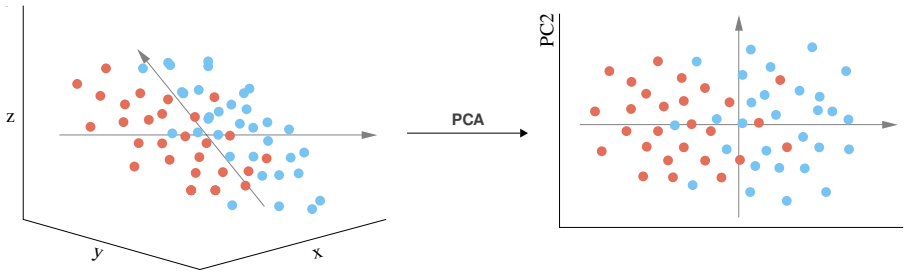


Figure 2.19: Process of PCA transformation applied to a dataset with with 3 dimension (x, y and z). The transformation allows to remove the least representative dimension and keep the two most diverse component ($PC1$ and $PC2$). The dataset has two classes (depending on the point color) in order to prove that PCA is an unsupervised process, which does not alter the relationships between classes.

Given a dataset $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n] \in \mathbb{R}^{d \times n}$, where $\mathbf{x}_i \in \mathbb{R}^d$ is the i-th data point, PCA is a linear dimensionality reduction technique that finds a low-dimensional representation of the data by projecting it onto a new orthogonal basis. The new basis is chosen to maximize the variance of the projected data, subject to the constraint that the basis vectors are orthogonal.

More specifically, let $\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_k] \in \mathbb{R}^{d \times k}$ be the matrix of k orthogonal basis vectors (i.e., $\mathbf{w}_i^T \mathbf{w} j = \delta ij$), where $k < d$. Then, the low-dimensional representation of the data is given by $\mathbf{Z} = \mathbf{W}^T \mathbf{X} \in \mathbb{R}^{k \times n}$. The columns of $\mathbf{Z}$ are the projected data points in the new k-dimensional space.

The PCA problem can be formulated as an optimization problem, where variance of the projected data has to be maximized, subject to the constraint that the basis vectors are orthogonal:

$$\max_{\mathbf{W}} \ \mathrm{Var}(\mathbf{W}^T \mathbf{X}) \quad \text{subject to} \quad \mathbf{W}^T \mathbf{W} = \mathbf{I}_k,$$

where $\mathrm{Var}(\mathbf{W}^T \mathbf{X})$ is the variance of the projected data and $\mathbf{I}_k$ is the $k \times k$ identity matrix. The solution to this problem is given by the eigenvectors of the covariance matrix $\mathbf{C} = \frac{1}{n} \mathbf{X} \mathbf{X}^T$, sorted in decreasing order of their corresponding eigenvalues. The eigenvectors are the basis vectors $\mathbf{w}_i$ and the eigenvalues are proportional to the variance of the projected data along the corresponding eigenvector.

PCA is utilized on a regular basis in a wide variety of applications including, but not limited to, image processing, biometrics, finance, bioinformatics, and engineering. In the context of anomaly detection, PCA can be utilized to identify unusual patterns in multivariate time series data through the process of comparing the reconstruction error of the original data to the reconstruction error of the reduced-dimensionality data. This is accomplished by comparing the original data to the reduced-dimensionality data [56].

In order to use PCA for anomaly detection [57], it must be applyed PCA to the initial multivariate time series data in order to acquire a set of principle components. This will allow us to use PCA for anomaly detection. According to the amount of variability in the data that each principle component explains, the principal components are ranked as follows: the first principal component explains the greatest variability, followed by the second principal component, which explains the second most, and so on. After that, the top k primary components that account for the majority of the data's variability can be selected, and then make use of those components to rebuild the initial data.

Upon the completion of the reconstruction, the reconstruction error is determined for each timestep by comparing the original data with the reconstructed data. Any appropriate distance measure, such as mean squared error (MSE) or Mahalanobis distance, can be utilized to compute the reconstruction error. Both of these distance measures are examples. A particular timestep is classified as abnormal if the reconstruction error for that timestep is greater than a threshold that has been previously established.

PCA can capture correlations between the variables in the data, which can be useful for detecting anomalies that are not visible in the individual variables. This is the primary advantage of using PCA for anomaly detection. Another advantage is that PCA can capture correlations between the variables in the data. In addition, it has the capability of lowering the dimensionality of the data, which can assist in the reduction of noise as well as the expenses associated with computing analysis.

AE are non-linear models, but PCA is a linear transformation technique. This is one of the most significant distinctions between the two. The data are projected onto a lower-dimensional subspace using PCA, which does this by locating a linear combination of the original characteristics that best reflects the variance in the data. AE, on the other hand, make use of a neural network to first learn a non-linear mapping from the input to a lower-dimensional latent space, and then they reconstruct the input from the latent space. This process is called autoencoding. AE are able to capture more complicated and non-linear correlations in the data as a result of this.

AE outperform PCA for detecting anomalies in datasets with complicated and nonlinear relationships. AE can learn more expressive representations of the data, resulting in improved reconstruction performance and more precise anomaly detection. Nevertheless, this comes at the expense of greater computing complexity and the requirement for vast quantities of training data.

In contrast, PCA is computationally efficient and may be applied to datasets containing fewer samples. With datasets with linear connections between features, it can also serve as a suitable baseline for anomaly detection. PCA may not capture all the information essential for successfully detecting anomalies in datasets with complicated nonlinear relationships.

2.4.8 Correlation Matrix

A correlation matrix is a table that displays the correlation coefficients that are present between different variables. In statistical analysis, correlation is a method for determining both the strength and the direction of a relationship between two variables. The range of possible values for correlation coefficients is from -1 to 1, with -1 indicating a perfect negative correlation, 0 indicating that there is no link, and 1 representing a perfect positive correlation.

In the process of data analysis, a correlation matrix is frequently used to gain a better understanding of the links that exist between the various variables that make up a dataset. It can be used to find patterns and trends, as well as variables that are highly associated to one another, and it can help identify patterns and trends. In domains such as finance, economics, and psychology, where researchers may be interested in understanding how several variables are related to one another, correlation matrices are a frequent tool that are utilized.

A correlation matrix is used to offer an overview of the relationships that exist between the variables that are contained in a dataset. Researchers are able to get insight into which variables are closely associated to each other and may then utilize this information to drive future analysis by examining the values in the correlation matrix. For instance, if a researcher is interested in predicting one variable based on another variable, they may use the correlation matrix to determine which factors are most closely associated to the outcome variable, and then use this knowledge to construct a predictive model.

The correlation matrix is computed using the correlation coefficient, also known as Pearson's correlation coefficient. The formula for the correlation coefficient between two variables X and Y, with sample size n, is:

$$r = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}}$$

where $\bar{X}$ and $\bar{Y}$ are the means of X and Y, respectively.

To compute the correlation matrix for a set of variables, the correlation coefficient between each pair of variables can be evaluated. For example, with variables X, Y, and Z, it is required to

calculate correlation coefficient between X and Y, X and Z, and Y and Z. The final correlation matrix might resemble something like this if it were created:

$$\begin{bmatrix} 1 & r_{X,Y} & r_{X,Z} \\ r_{Y,X} & 1 & r_{Y,Z} \\ r_{Z,X} & r_{Z,Y} & 1 \end{bmatrix}$$

where $r_{X,Y}$ is the correlation coefficient between X and Y, $r_{X,Z}$ is the correlation coefficient between X and Z, and so on.

2.4.9 Metrics

Metrics are functions used to monitor and evaluate a model's performance. The objective of the metrics is to quantify the performance of the model in the prediction task. The most commonly used evaluation metrics for regression problems are Root Mean Squared Error (RMSE), Mean Squared Error (MSE), Mean Absolute Error (MAE), R^2.

- Mean absolute error (MAE) represents the average of the absolute difference between the original and predicted values in a data set.
- Mean Squared Error (MSE) represents the average of the squared difference between the actual and predicted values in a data set.
- Root Mean Squared Error (RMSE) is the square root of the Mean Squared error. The advantage over MSE is that the error is comparable with the output measure. It measures the standard deviation of the error, so it is the most widely used.
- R^2 is an indicator that, starting from the regression line, summarizes in a single numerical value how much the analyzed quantity deviates from that line on average, so it is an indicator of model goodness-of-fit. The best performance is achieved when this value is equal to 1.

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}|$$

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2$$

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2}$$

$$R^2 = 1 - \frac{\sum(y_i - \hat{y})^2}{\sum(y_i - \bar{y})^2}$$

where:
$\hat{y}$ - predicted value of y
$\bar{y}$ - mean value of y

The less MAE, MSE, and RSME values are, the better a model fits the data. The greater the $R2$ value, the better the model fits a data set and the better the predictor variables can explain the variance in the response variable.

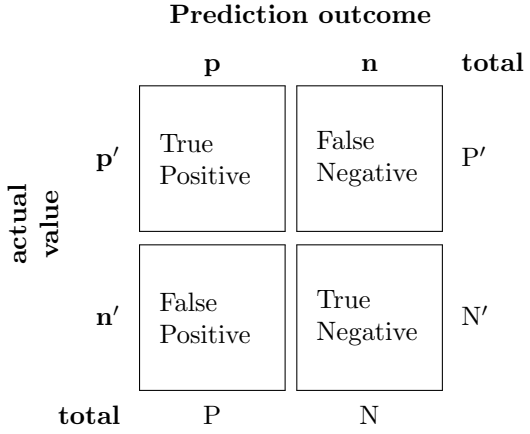
Since they are based on the square of the error, RMSE and MSE are more sensitive to observations that are further from the mean than MAE. This suggests that RMSE is most beneficial when huge mistakes are very undesirable.

Each epoch's performance indicators are measured and then shown as learning curves. The best aim is for the training loss and validation loss to be as near to each other as feasible, while both are close to 0.

Confusion Matrix

A confusion matrix is a table that summarizes a classification model's performance on a set of test data. It is a matrix with rows and columns matching to the true and anticipated classes, respectively. Each column of the matrix reflects the number of test samples belonging to a certain true class that were classified as a specific predicted class. The goal of the confusion matrix is to offer a quantitative evaluation of the performance of the classification model.

Suppose, having a binary classification problem with two classes, "Positive" and "Negative", in which predictions are made on a test set using a machine learning model. The confusion matrix can be represented as follows:



Using the values from this confusion matrix, various metrics can be calculated such as accuracy, precision, recall, F1-score, and others. Formally, those metrics can be written as:

Accuracy:

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

Precision:

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

Recall (also called Sensitivity or True Positive Rate):

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

F1-score:

$$F1\text{-}score = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

Specificity (also called True Negative Rate):

$$Specificity = \frac{TN}{TN + FP}$$

False Positive Rate:

$$False\ Positive\ Rate = \frac{FP}{TN + FP}$$

False Negative Rate:

$$False\ Negative\ Rate = \frac{FN}{TP + FN}$$

Moreover, the confusion matrix may be utilized to discover certain sorts of model mistakes, such as false positives and false negatives. This information can aid in finding improvement areas and refining the parameters of the model.

ROC Curve

A *Receiver Operating Characteristic* (ROC) curve is a graphical depiction of the performance of a binary classifier system as the discrimination threshold changes. The graph compares the *true positive rate* (TPR) to the *false positive rate* (FPR) for different threshold values. The curve gives a visual depiction of the trade-off between the true positive rate and false positive rate for a classifier and allows for a quick and easy method of evaluating a model effectiveness. *Area Under the Curve* (AUC) is a statistic generated from the ROC curve that summarizes the performance of the classifier in a single scalar number [58].

The confusion matrix provides the raw data used to construct the ROC curve, while the ROC curve provides a graphical representation of the performance of a binary classification model based on different threshold values.

Pros of ROC Curves:

- ROC Curves give a clear visual depiction of the trade-off between the true positive rate and false positive rate of a classifier, allowing for a straightforward comparison of several models.
- Using the ROC curve, the optimal threshold for a classifier may be determined by maximizing the false positive rate and the true positive rate.
- The Area Under the Curve (AUC) measure obtained from the Receiver Operating Characteristic (ROC) curve offers a single scalar number that represents the overall performance of the classifier.
- ROC Curves are utilized extensively in machine learning and may be applied to a variety of binary classification issues.

Cons to ROC Curves:

- ROC Curves are not appropriate for multiclass classification tasks and must be modified for usage in these circumstances.
- The ROC curve can be susceptible to changes in the distribution of the underlying data and may not always accurately represent the performance of a classifier.
- Interpreting the ROC curve might be challenging for those with insufficient statistical knowledge.
- The ROC curve does not give information on the precision or recall of a classifier; hence, other metrics may be required to measure its performance in its entirety.

2.4.10 Normalization

Normalization is a data preparation method used to adjust the values of a variable to a certain range. This is done to enhance the performance of machine learning algorithms and make the data more consistent and easier to interpret..

Min-Max Scaling and *Standardization* are two typical normalizing techniques:

- **Min-Max Scaling** entails transforming the data into an interval between 0 and 1. This is accomplished by removing the variable's minimal value and dividing by its range (maximum value minus minimum value). For instance, if a dataset contains ages ranging from 20 to 60, Min-Max Scaling may be used to change the numbers to a range of 0 to 1.

$$X_{sc} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

- **Standardization** entails altering the data such that the mean is 0 and the standard deviation is 1. This is accomplished by removing the variable's mean value and dividing by its standard deviation.

$$z = \frac{x_i - \mu}{\sigma}$$

Normalization is used in machine learning algorithms since it may lessen the influence of outliers and enhance the performance of particular algorithms. Many machine learning methods assume that input data is regularly distributed, and It assists in achieving this assumption. *Normalization* may also enhance the interpretability of the data by making it simpler to compare and comprehend the variables within a dataset.

The machine learning techniques k-nearest neighbors, support vector machines, and neural networks all benefit from *normalization* [59]. These methods analyze the similarity between data points using distance metrics, and it helps to guarantee that the distance metrics are consistent across all variables. Moreover, when input data is normalized, some algorithms, such as neural networks, may converge more quickly.

Chapter 3

Methodology

This section lists and explains the different procedures used in the implementation of the process.

3.1 Introduction

The purpose of this thesis is to define a data-driven framework for condition-based maintenance. An analysis of the WTGs performances is firstly performed, the ultimate product will be an algorithm that returns metrics for understanding the quality of the turbine and its subsystems throughout operation. The generated metrics should be able to measure and differentiate occurrences associated with system faults against instances of optimal operation. Therefore, the developed model should be able to show malfunctions related to the different turbine subsystems.

The development of the algorithm will demand the availability of vast quantities of data and will not be restricted to the examination of single turbines. The model will be trained using an unsupervised anomaly detection approach. This choice depends on the fact that it is not possible to have large amounts of correctly labeled data, in which a malfunction occurs on a particular turbine sub-component. However, through various performance metrics, is possible to limit the unwanted behaviors that make up a large portion of the available data, in which anomalies are also contained. By examining and analyzing the problem for a subset of outstanding candidates, the method must be able to yield good results when applied to additional candidates as well. In addition to evaluating turbines that represent distinct instances of the same model, blended methods will also be examined. With these approach, it will be feasible to assess algorithms trained on certain model types but tested on others.

Using several classification-related test and metric, the algorithms functionality will be validated. It will be assessed based on their capacity to distinguish between several significant events associated with the operation of WTGs in real-world scenarios. But, they will also be assessed using benchmarks that were generated intentionally by inserting outliers and other abnormalities into the temporal data.

3.2 Data Selection and Filtering

During the data selection step, several turbines are assessed based on a variety of attributes for subsequent usage in the model training and testing stages. Remember that autoencoder models for anomaly detection are driven by high-performance data to enable accurate identification of unconventional data patterns during testing and eventual production usage.
Thus, the aim of this step is to choose a subset of turbines with sufficient data to provide a proper training sample, as well as filtering those data in order to provide good training samples in the consequent training phase. During the filtering process different KPIs are involved, such as performance evaluation, energy production, downtimes and other measures.

3.2.1 Performance Filtering

The performance of the WTG can be calculated as the ratio of produced power to producible power at each time period:

$$Performance = \frac{P}{P_p}$$

This performance metric can be used to identify any periods of time when the WTG is not performing as expected, such as when the performance drops below a certain threshold, indicating a potential issue with the WTG or the wind conditions.

Formally, the performance at a given time t, based on the active power output P_{act} and the maximum producible power P_{max} at the corresponding wind speed, can be expressed as:

$$\text{Performance}(t) = \frac{P_{act}(t)}{P_{max}(v(t))}$$

where $v(t)$ is the wind speed at time t and $P_{max}(v(t))$ is the maximum producible power at that wind speed, as determined by the power curve of the WTG.

3.2.2 Temperature Filtering

Filtering data in a WTG by examining the absolute temperature differential between the gearbox oil and the gearbox bearing temperature may be utilized to discover possible problems or anomalies in the functioning of the WTG. If the temperature differential is too great, it may signal that the gearbox is not performing as intended, which may result in greater wear and tear, decreased efficiency, and the possibility of catastrophic failure. Formally, the difference in temperature between the gearbox oil and the gearbox bearing temperature can be expressed as:

$$\Delta T = T_{oil} - T_{bearing}$$

where ΔT is the temperature difference, T_{oil} is the temperature of the gearbox oil, and $T_{bearing}$ is the temperature of the gearbox bearing. If the temperature difference ΔT is too high, it can lead to a number of issues, such as increased friction between the gearbox components, increased wear and tear on the gears and bearings, and reduced efficiency of the WTG. In extreme cases, a high temperature difference can lead to catastrophic failure of the gearbox, which can result in costly repairs and extended downtimes for the WTG.

For this reason, by observing those temperatures, it is possible to detect potential issues with the WTG before they become more serious, and to take proactive actions to address them, such as conducting maintenance or replacing faulty components.

3.2.3 Status Filtering

As well as *SCADA* data related to sensor measurements, WTGs also have a number of components aimed at observing and reporting the presence of states related to their operation. These statuses may be roughly classified into three categories:

- **Ok status**: characterizes an operational turbine that is producing power.
- **Idle status**: Identifies a turbine that, although it is not malfunctioning, is not producing power, either due to external causes (lack of wind) or due to demand for shutdown or main cable unrolling.
- **Error statuses**: Signalling system malfunctions and stops.

Furthermore, the existence of status readings due to an alert, defect, or communication problem is cause for concern and may contradict the notion of healthy behavior. For this reason, the healthy turbine operation data do not include timestamps characterized by the alarm states. In contrast,

there is no need to eliminate idle states because they do not in themselves constitute malfunctions; nonetheless, these states may be characterized by the presence of other numbers associated with abnormal measurements. In conclusion, most of the idle states are indirectly removed using other filters. While these actions are frequently adequate to remove the majority of outliers, further considerations may be made by studying the remaining values of the wind turbines' active power and other measurements in more detail.

3.2.4 Limitations Filtering

The power restrictions may be the result of maintenance requirements or limits imposed by *Terna S.p.A.*, the Italian Transmission System Operator (TSO), which is in charge of managing the balance between energy demand and offer in real time. At times, a WPP turbines may need to operate in low-power mode and send less electricity into the grid. In these moments, the turbines exhibit unexpected behavior; hence, the impacted samples may be deleted. This information is not encoded with a T-Status code and has to be pulled from a company-supplied supplemental data collection.

3.2.5 Other Filters

Other filters were included in the process along with the ones previously considered. They directly fix upper and lower limits to some of the observed quantities, such as active power and wind speed with the aim of removing outliers and corrupted data.

3.3 Data Preprocessing

The previously filtered data was then modified for use in model training and testing. The models are always fed sequential data, therefore each sample used for training and testing is always comprised of a series of distinct timesteps, each of which consists of the observed variables. The data, including training and test splits, were processed using a conventional scaler throughout the data preparation step. Scaler if fitted on training data and then use consequently on all the testing data using a particular model. The subsequent step consists of generating time sequences from data in non-sequential format.

It should be considered that training data and test data differ significantly:

- **Training data**: Consists of a set of sequences from different turbines, not equally participating in the final composition. Each sequence in this dataset is characterized by having every timestamp of every sequence belonging to the anomaly-free data cluster. Each individual filter, in fact, grouped in a logical OR operation (positive labels equals anomalous) allows the isolation of sequences uniquely consisting of ideal data (the ones with negative labels). In this way, the training set can be significantly smaller than the entire dataset, as it is stripped of all sequences that contain at least one anomaly timestamp. To overcome this problem of data scarcity, some filtering operations are performed by exploiting metrics in terms of moving average (with strict filters) and with time single timestamps (with more loose filters).

- **Test data**: Test data is not filtered as it is necessary to discriminate the data in detection, however, the considerations made regarding filters will instead go to characterize the anomaly labels associated with the test data. These labels, produced considering different indicators and different metrics during the various tests, will be specifically discussed.

3.4 Models Processing

3.4.1 Models Training and Tuning

A number of potential models will be developed to recreate time sequences throughout the training phase. All neural network models used for this task are AE. As described in the preceding sections, the goal of this architecture is to create a compressed (latent) representation of the observed data by attempting to rebuild the data such that it closely resembles the original. During training, model parameters are optimized using losses that measure the inaccuracy created after each reconstruction of a batch of data.

The algorithms' hyperparameters will then be used to examine the various models and architectures. The main AE flavours include:

- Feed forward AE: Basically a multi-layer perceptron.
- Recurrent AE: Uses LSTMs and GRUs as recurrent models.
- Variational AE: AE Uses a Gaussian probability distribution in the latent space.
- Denoise AE: AE used in denoising task instead of a data reconstruction.

It should be noted that the different types of AE will be used in combination therefore the models will have many variations.

Several training parameters will be used during the training phase, some of which are: the batch size, the optimizers and number of epochs. Algorithms with a higher degree of regularization will be trained on more epochs to produce an effective level of training therefore it will last as long as an appropriate level of loss is reached.

3.4.2 Models Validation

To ensure that the models are not overfit, a validation set will be used throughout the training phase. The validation set will only be used during the reconstruction step and not during following stages of anomaly detection. Other techniques will be used during the actual testing phase of anomaly detection, which led to this decision. Thus, these various model assessments in the training phase will attempt to reflect validation or testing models. The validation dataset is obtained by selecting a random subset of sequences from the training data, with a distribution of 1 validation data versus 9 training data.

3.5 Model Evaluation

Once the model has been trained and validated on the reconstruction task. The algorithms will be tested on different anomaly detection tasks, which consist of identifying different events in the turbine time series.

Experiments to test the effectiveness of the model will resort to the study of various phenomena related to real data, but also to the verification of detection of artificial anomalies injected into original *SCADA* data.

The detection of anomalies can be distinguished into two stages, as shown in Figure 3.1, the procedure is addressed partially in this thesis.

Offline anomaly detection, also known as batch processing, involves analyzing historical data to identify anomalies. The data is collected and stored over a period of time, then analyzed in batches to identify patterns and anomalies. The output of this process is a set of rules or models that can be used to detect anomalies in real-time.

On the other hand, online anomaly detection involves analyzing data in real-time as it is generated. This method is used when it is critical to identify anomalies as soon as they occur, such as in a production system or in network traffic monitoring. In online anomaly detection, the system continuously collects and analyzes data in real-time, looking for deviations from normal behavior. When an anomaly is detected, an alert is triggered so that appropriate action can be taken.

In this thesis, the process is similar, however the step of evaluating the residuals is only considered in part, in favor of measuring the performance of the various models based on ROC curves. If the residuals evaluation approach were considered in its entirety, the measurement in the testing step would consist only of a classification result. However, the approach described in this paper is not reduced to the determination of a threshold for the purpose of evaluating these events, but claims to consider more well-grounded aspects related to the study of anomalies.

The approaches noted in the study can then be used to extract metrics designed to represent thresholds for alarms. However, in the context considered, alarm events are not sufficient in number to accurately determine these thresholds without also including events that are not of interest but still anomalous.

3.5.1 Status and Filter Discrimination

The objective of anomaly detection in machine learning is to find odd patterns or events within a given dataset. In many instances, the dataset consists of examples that have been tagged as either normal or abnormal. For this reason, the first test involves a task in which the filters used for finding the right training sequence are employed again in a labelling process. Secondly, the model is used in a task of *partial testing* for the Error and Inactivity Status.

During this stage, the MSE will be used to aggregate the residuals produced by the difference between the actual measurements and the reconstructed measurements over all of the different magnitudes and all of the different timesteps. The time sequences that contain only one timestep that is reported as being abnormal will be eliminated from the dataset in an effort to cut down on the number of borderline situations.

In anomaly detection, *partial testing* is used to evaluate the performance of a machine learning model on a subset of the training classes. This means that, during testing, the model is assessed on a subset of the classes used for training and not the complete set.

There are various reasons why anomaly detection use partial testing. One reason is that it can provide a more accurate evaluation of the performance of the model. In the majority of real-world situations, anomalous events are uncommon and the great bulk of data is typical. So, it is essential to evaluate the model's capacity to detect anomalies in a realistic scenario, where the model is exposed to a big quantity of normal data and a small quantity of abnormal data. A further reason why partial testing is utilized in anomaly identification is that it can help determine the model's strengths and shortcomings. By testing the model's performance on a subset of the classes used for training, it is possible to determine the classes on which the model excels and those on which it struggles. This can assist guide future model enhancements. Randomly separating the labeled data into training and testing sets is a popular way to partial testing in anomaly identification. Training set is used to train the model, while testing set, is used to evaluate its performance. During testing, however, just a subset of the conditions utilized during training are utilized. This method is frequently employed in unsupervised anomaly detection, where the objective is to discover unusual events without using labeled data.

3.5.2 Benchmark Evaluation

In the evaluation through benchmarks, the different models will be used in the detection of particular anomalies artificially generated through the injection of noise or transformation of certain measurements associated with particular subsystems. Some of these transformations come from the scientific literature and their nature is well documented [60].

The realization of these benchmarks will consist first in the creation of an altered signal by introducing modifications by starting from the original. Then, using a binary mask computed with probabilistic approaches, timestamps will be replaced in the original time sequences. The masks will form the labels of the anomalous events to be identified in the testing phase of the model. However, given the chaotic nature of the WTG systems, it will first be necessary to ensure that the original signal is free of previous anomalies. If it is not, there is a risk of introducing anomalies into a time series that already has additional unreported anomalies. This would cause a lowering of the performance of the model during testing, since previously present anomalies would not be reported as such [61]. In the observed literature, the duration of these phenomena is extremely short (on the order of a few minutes) therefore, the identification of these discontinuities will be limited to the study of individual timesteps unlike previous tests. In the observed literature, the duration of these phenomena is extremely short (on the order of a few minutes) therefore, the identification of these discontinuities will be limited to the study of individual timesteps unlike previous tests.

The nature of this experiment differs considerably from a test performed by finding faults or alarms on real data: the main difference lies in the fact that such events can occur involving several subsystems, even if only one alarm may be triggered on the subsystem in the most critical state.

In spite of their different nature, this experiment is nevertheless interesting for verifying the degree to which different observed quantities turn out to be correlated with each other in the training phases. Indeed, by observing the reconstruction error on the individual quantities, one can verify how the model reacts on the others to an anomaly concentrated on one sensor.

This experiment is designed to be a departure from what has been considered in the past. If it turns out that in the event discrimination, the sequences come aggregated interametne among all of the timesteps and all of the measures, then in this experiment, the aggregation with MSE will work only on the measures, leaving the timesteps untouched. For this reason, the experiment will not only have to do with finding whether or not there are anomalies throughout the sequence, but it will also have to do with determining where the anomaly is located inside the issue that is being considered.

3.5.3 Downtime Evaluation

In the process of analyzing AE models, the very last phase is the investigation of downtimes. The WTG downtimes that are being considered are made up of a sequence of alarms that go off whenever a given quantity reaches a predetermined upper or lower limit in relation to that amount. The measurements taken at this point are approximately comparable to those taken earlier. In point of fact, downtimes are already recorded along with the relevant alarm state. As a result, the detection of them involves making use of the work that was done before in the event discrimination steps.

On the other hand, the study of the individual quantities that are present in the environment during these downtimes is the most essential part of these events. After that, this evaluation will take place, both qualitatively and statistically, with the goal of visualizing, using the metrics that have previously been presented, the kind of event that will take place. However, the study of the individual quantities will be the single most significant part, as this will allow researchers to attempt to have an accurate picture of what these phenomena are and the corresponding countermeasures that are targeted at minimizing them. Following that step, particular models will be chosen for commentary based on how well they explain the observed phenomenon.

The occurrence of these downtimes is exceedingly seldom, and in each of the clusters, there were at most a handful of turbines that exhibited a given category of event. Their nature may also be easily grasped in a very short amount of time due to the fact that, despite the fact that only one quantity might be to blame for these occurrences, all measures are able to recognize departures from these in terms of reconstruction error. So, it is necessary to do this study on an individual basis for each downtime in order to get a proper understanding of what occurred and whether or not the models can actually recognize it.

Ultimately, in this experiment, the labels that were obtained with the error states and the timestemps that were defined by inactivity will be used to have both a qualitative and quantitative indicator of the capability to detect fault statuses.

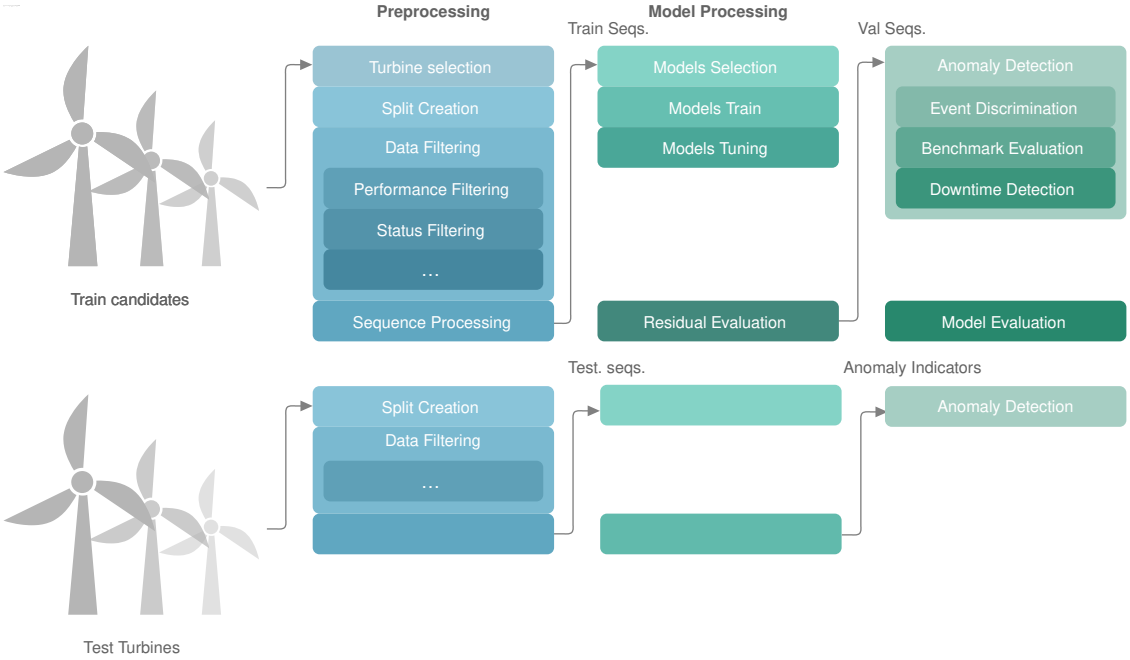


Figure 3.1: Flowchart of the fault detection using online and ofline phase.

3.6 Performance Analysis

The algorithm can be used to verify the real performance of the turbines and reconnect them to the various subsystems of the devices once the best models have been determined through the various anomaly detection tests.

For this reason, at the end of the experiment, there will be a study that is specifically called *Performance Analysis*, where an entire set of turbines will be observed in order to understand and verify that observed deviations in magnitudes are somehow related to the macroscopic functioning of these energy systems. This study will be conducted in order to understand and verify that observed deviations in magnitudes are somehow related to the macroscopic functioning of these energy systems.

This aspect of the work contributes significantly to the validation of models as well as the analysis of wind farms. In this part of the article, a number of graphs and metrics will be provided that can subsequently be utilized as an example of how the method should be applied.

Chapter 4

Experiments

4.1 Gamesa G90

The next experiments involves 9 turbines G90 in the a single wind farm in southern Italy. The main technical specifications of the G90 wind turbine are as follows:

- Power:
  - Rated power: 2.00 MW.
  - Cut-in wind speed: 3 m/s.
  - Cut-out wind speed: 25 m/s.
  - Survival wind speed: 49 m/s
- Rotor:
  - Blade number: 3.
  - Diameter: 90 m.
  - Area: 6,362 m^2.
- Gearbox:
  - Type: spur/planetary.
  - Gear stages: 3.
  - Transmission Ratio: 0.101.
  - Brand: Echesa (Gamesa Group)/Hansen/Bosch Rexroth/Winergy.
- Generator:
  - Type: Doubly-fed Asynchronous.
  - Brand: Cantarey.
  - Voltage: 690 V.
  - Main frequency: 50 Hz.
  - Speed max: 1900 U/min.
- Weight:
  - Single Blade: 5.8 t.
  - Hub: 18.6 t.
  - Rotor: 36 t.
  - Nacelle: 70 t.
  - Tower: 255 t.
  - Total: 361 t.

4.2 Data Selection and Preprocessing

At this stage of the work, a set of G90 turbines available for the experiment will be partitioned and studied to allow suitable and usable data to be produced for subsequent training and testing phases.

4.2.1 Data Selection

In the data selection step a set for WTGs is considered in a particular plant among several available. In our experiment on G90s the subset of turbines will then be divided into 2 groups:

- **High-performance (Training/Validation) turbines**: These turbines will be used to train effectively the models used for anomaly detection. These turbines will then also be used for a subsequent validation and testing phase. For this reason, the data proceeds from these turbines will be divided into several splits used separately in the training and testing phases. Training and test splits will prevent overfitting issues and let the algorithm to be assessed when applied to turbines used in training. Due of the vast number of similar wind turbines in the same location, it was decided to train neural networks utilizing a group of unique wind turbines. This cluster is made with 4 turbines, these are called: *WTG-1*, *WTG-2*, *WTG-3* and *WTG-4*.
- **Other (Testing Turbines) turbines**: These turbines will be used exclusively in the testing phase to verify the behavior of the algorithm on data proceeds from unobserved turbines (in the training phase). This test will allow the models to be evaluated on new data with some degree of diversity from previous data. This whirlwind there are 4 and will be called: This cluster is made with 4 turbines, these are called: *WTG-5*, *WTG-6*, *WTG-7*, *WTG-8* and *WTG-9*.

The decision to combine the data from many turbines in the training phase, was dictated by the lack of extensive training periods, since the data from a single turbine was inadequate to train neural networks adequately. In addition, this decision makes it feasible to avoid neural networks from learning to detect the operation of a single turbine correctly, and provides a good degree of generalization for testing on unseen data with good performance.

Specified data for both high-performance and other turbines will be provided for one year. With the difference, however, that in the case of high-performance turbines part of the period will be used in the training phase. To have an appropriate degree of diversity in the data provided in the training phase, these periods will be selected in different parts of the year. While a great deal of data has been made public, only the last year is available. This amount of data may not be sufficient for training neural networks if seasonal phenomena have to be included. Specifically, the year will be split into three four-month periods, and each of these will be subdivided into three splits (splits shown in Table 4.1):

- **Training split**: For the first 3 months of the four-month period, after the filtering process described next.
- **Validation split**: Random samples from the previous set.
- **Testing split**: Consists of the last month in the four-month period.

4.2.2 Dataset Measures

It is essential, while picking characteristics for anomaly detection, to choose those that are pertinent to the particular issue you are attempting to address and have a high signal-to-noise ratio. In certain instances, it may be necessary to contain all of the traits, while in others, a subset of the qualities may work. It is essential to use domain expertise and data analysis to establish the

Period	Split	N° Timestamps
Jan, Feb, Mar	Train+Val	12960
Apr	Test	4319
May, June, July	Train+Val	13248
Aug	Test	4463
Sep, Oct, Nov	Train+Val	13248
Dec	Test	4463
	Train Total	**39456**
	Test Total	**13245**

Table 4.1: Table representing the different splits in the experiment and the corresponding size of the datasets.

optimal collection of characteristics for a specific situation.
Therefore, the *SCADA* measurements considered in the reconstruction and anomaly detection phase will be:

- Active Power.
- Blade Pitch Angle.
- Gearbox Bearing Temperature.
- Gearbox Oil Temperature.
- Generator RPM.
- Generator Temperature.
- Wind Speed.
- Reactive Power.
- Rotor RPM.

These measures are all characterized by being 10-minutes data obtained through 10-minute average statistics. Generally, this method is performed to reduce the amount of storage space required for the vast quantity of data that is continuously generated by wind farms. The quantities can be grouped into several categories to identify the most important subsystems of the WTGs:

- **Gearbox subsystem**: Considered from a mechanical point of view, such as *Rotor RPM* and *Generator RPM*), and also from a thermal point of view *Oil Temperature* and *Bearing Temperature*.
- **Rotor subsystem**: Measures such as *Rotor RPM* and *Wind Speed*.
- **Generator subsystem**: Quantities like *Generator RPM*, *Generator Temperature*, *Active Power* and *Reactive Power*.

Along with these metrics in the next phase of data filtering and dataset creation other metrics will be used, either obtained from additional turbine *SCADA* sensors or obtained by transforming previously collected measurements. These additional measures will be used in an attempt to detect possible breakdowns and to filter anomalous events in the data for the training.

4.2.3 Data Filtering

After the extraction of data for each individual turbine, the data was filtered. The filters used are described in Table 4.2. These include both filters on individual quantities and filters obtained by considering other metrics:

- **Filters on dataset metrics**: Such as *Active Power* and *Wind Speed*. That allow the moments associated with low turbine operativity to be removed.
- **Filters on additional metrics**: Such as *Perfomance*, *Difference in gearbox temperatures*, *Status* and *Limitations*. These remove moments associated with irregular work and reduced productivity.

Measure	**Lower Bound**	**Upper Bound**	**Other Conditions**
Active Power	-50kW	2300kW	
Wind Speed	3.5m/s	25m/s	
Performance (G90)	50%	200%	
ABS Temp. Difference		10C°	
Status			Not error status
Perf. (G90) Cent. MA 12	80%	150%	
Limitations			Not a limitation

Table 4.2: Types of filters used in the filtering phase. The table shows the list of quantities used and the appropriate filter conditions. All data less than the lower bound or greater than the upper bound were removed. Since *Status* is a symbolic data type the *Error Status* is removed.

Filters used directly on the dataset quantities make it possible to eliminate negative active power or excessively high values of it, probably the result of sensor fluctuations. Another important discrimination measure is wind speed. This procedure also excludes any samples in which the wind speed is below the cut-in speed, below which the rotor cannot operate. This procedure also excludes any samples in which the wind speed is below the cut-in speed, below which the rotor cannot operate.

As previously announced in the Methodology section, performance metrics allow for a discrepancy between the energy produced and the energy that can be produced by the WTG system. Therefore, by considering the power curve of the G90 model in question, it is possible to estimate the *producible energy* as *Wind Speed* changes and compare it with the measured *Active Power*.

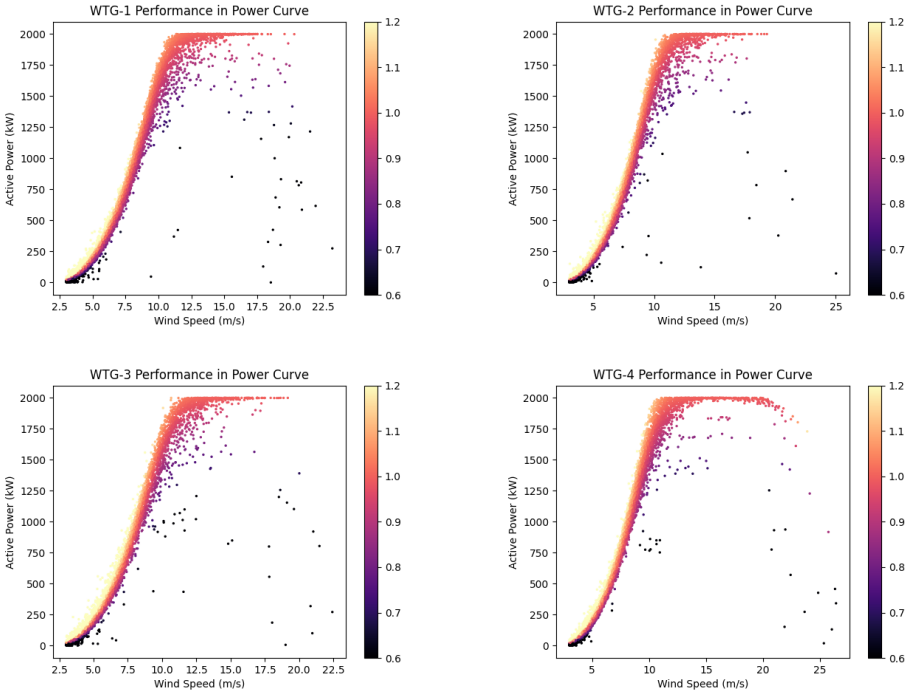


Table 4.3: Cross plots of the performance from the WTGs involved in training. The color of the points represents the performance value.

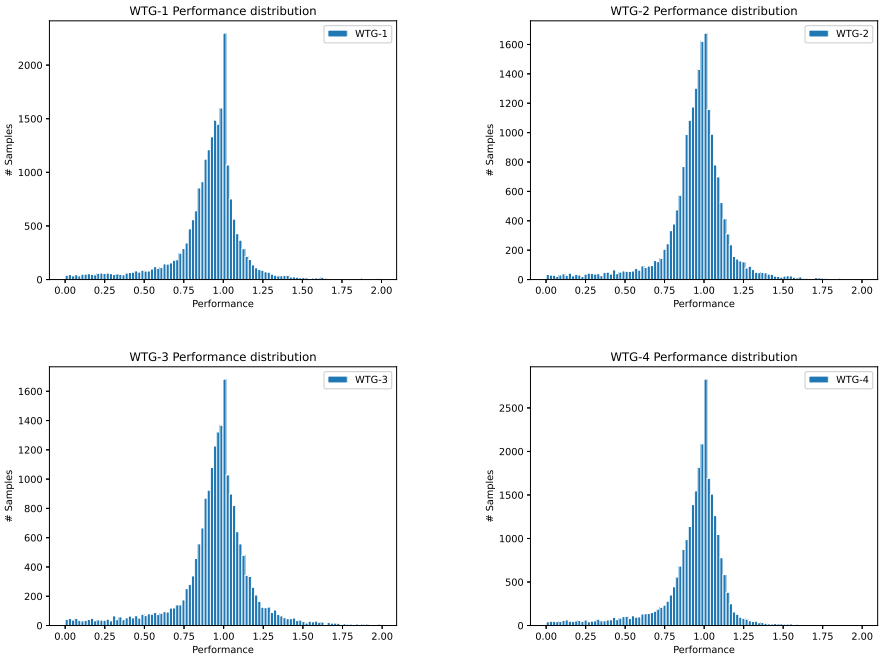


Table 4.4: Performance distributions from the WTGs involved in training.

In Table 4.3, there are figures showing SCADA data of turbines observed in scatter plots, Wind

Speed-Active Power. I graph shows each data with color mapped against the relative performance value. In this way it is possible to see how the data underlie the shape of the curve and verify the variability of the observed performance.

The different performances are then shown in the histograms in Table 4.4. It is evident from the table that the data have a normal distribution with a center around 1. This is not a random number, since the training turbines were selected according to their high productivity. Additional turbines associated with subsequent tests will have data distributions with lower mean values.

Performance metric is used as filter both on instantaneous data (directly on 10 minutes data) but also on sequential ones (using MA with 12 periods). Using only the instantaneous data would cause too sharp an evaluation of the data by excluding too many of them; MA, on the other hand, allows a data to be evaluated in an adjacent set of values, achieving a harmonizing effect.

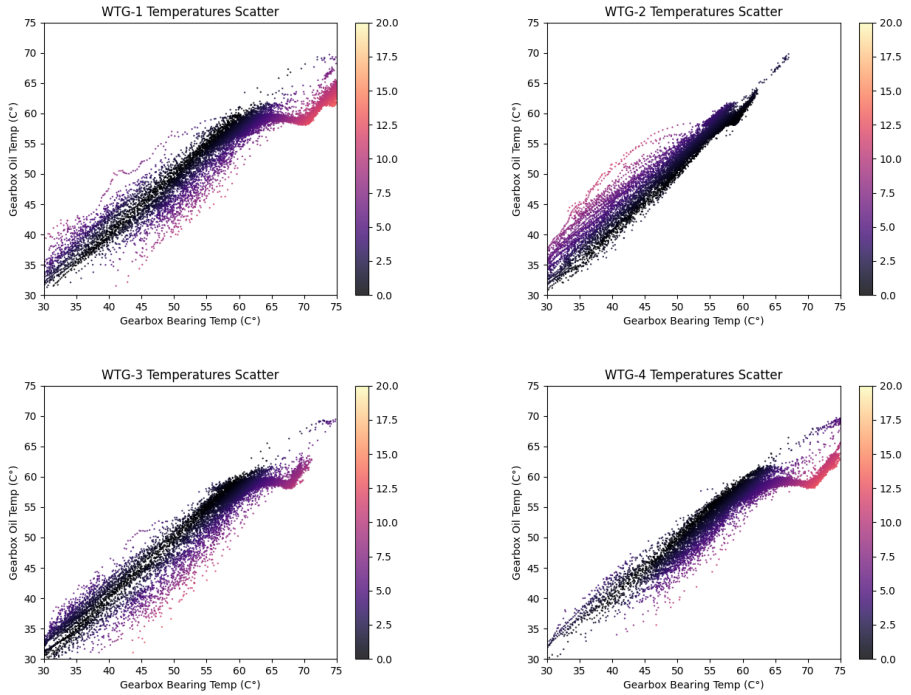


Table 4.5: Cross plot of the gearbox oil and bearing temperatures from the WTGs involved in training. The color of the points represents the absolute value of the difference between the two temperatures.

In addition to metrics related to the productivity of the individual turbines, which are mainly useful for verifying the actual operation of the turbines. A metric related to gearbox operation from a thermal point of view is also considered. The temperature difference between the oil and the bearing of the latter is shown in images of Table 4.5. In the filter operation the difference is measured in absolute value is compared with a constant difference, higher values are discarded. This decision of hangs on the fact that turbines with high temperature difference have irregularities in the cooling fluids.

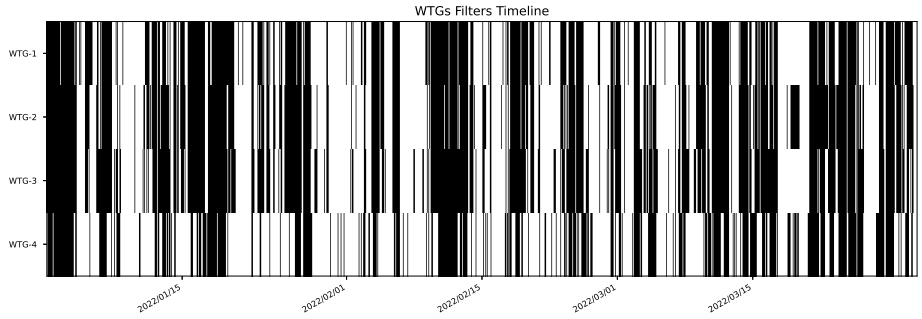


Figure 4.1: Comparison of the filtered timeline in all the training WTGs. Black bands represents filtered timestamp while the white ones are the one kept for training. It is evident, how often, data removed from timelines are aligned between various turbines.

All previously considered filters are finally encapsulated through a logical-and operations into a single label indicating the validity of the timestamp in training. A candidate sequence for training becomes such only if all the data in it are characterized by being valid. In Table 4.6 the logical-and operation can be observed in the WTG-1 dataset. The overall result allows us to limit the data in use as a small number of training sequences. It can be observed how much of the sequences are filtered by multiple conditions simultaneously, a symbol of how non-ideal data can be found by multiple indicators. Another interesting aspect is how, inactivity status data are still almost entirely excluded from the other metrics although they have not been filtered.

The power restrictions may be the result of maintenance requirements or limits imposed by the TSO. At times, a wind farm's turbines may need to operate in reduced power mode and send less electricity into the grid. At these times, the turbines exhibit unexpected behavior; hence, the impacted samples may be deleted. This information is not encoded with a Status code and has to be pulled from a company-supplied supplemental data collection.

Figure 4.1 shows the overall validity labels in the datasets of the different training turbines. The finding that clearly emerges is how in many instances observed the data on invalid states are shared by all turbines, probably because they are the result of environmental causes (periods of inactivity).

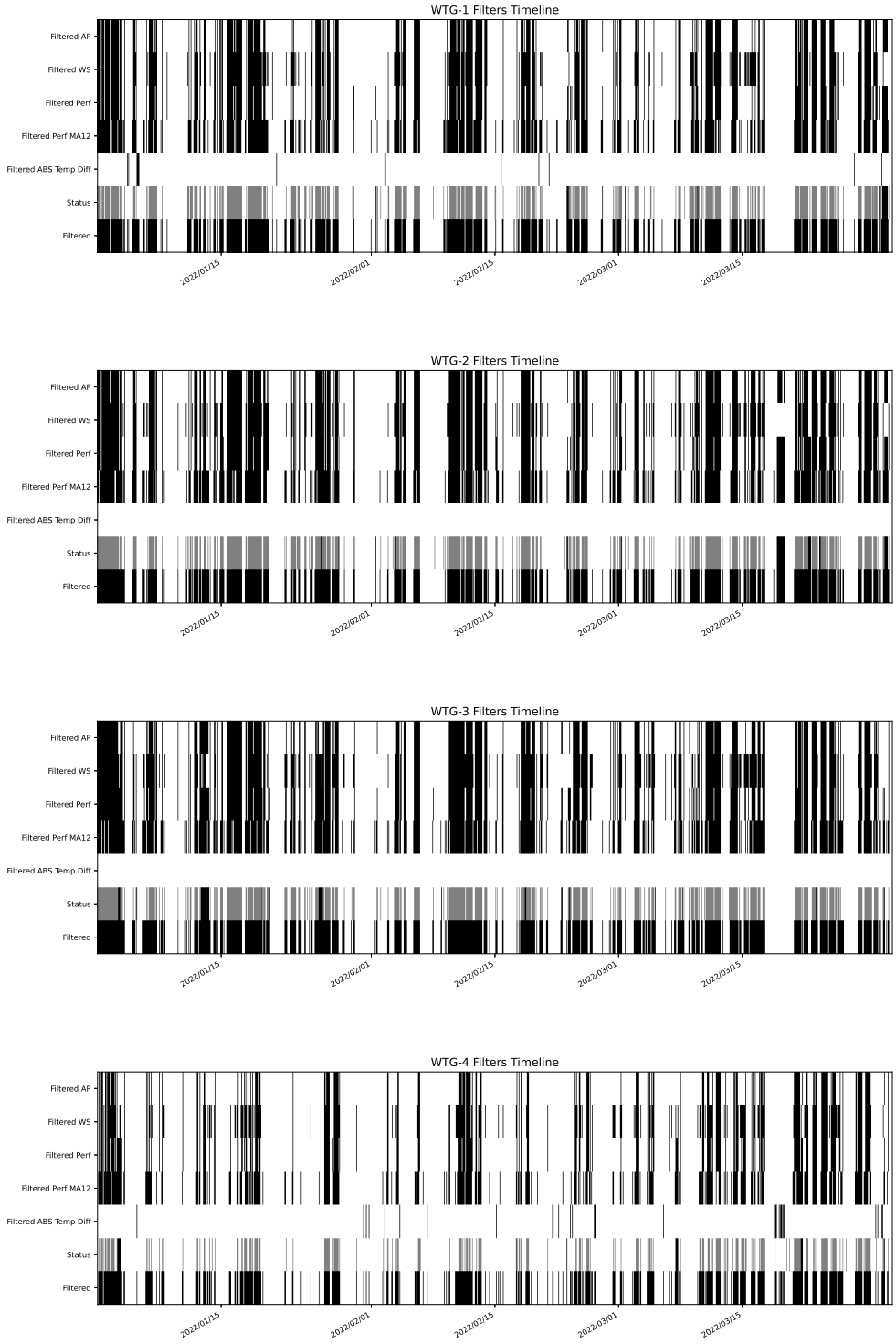


Table 4.6: Filtering process displayed in a band plot for the first training period (january, february and march) for the training WTGs. The last band plot is the logical-and between all the previous filters. Black stands for removed timestamps. The gray color in *Status* plot band stands for a *Inactive Status*, while black represents the *Error Status*.

4.2.4 Feature Selection

When selecting features for an anomaly detection job using machine learning, it is essential to examine the connection between the features. A correlation matrix may be used to find strongly associated characteristics, which can lead to issues such as:

- **Redundancy**: Strongly correlated characteristics may supply duplicate information, which may raise the model's complexity and make it more difficult to comprehend.
- **Overfitting**: If strongly correlated characteristics are included in the model, this might result in , where the model fits the noise in the data instead of the underlying pattern.
- **Instability**: Adding strongly correlated characteristics might render the model unstable, resulting in substantial output changes for modest input changes.

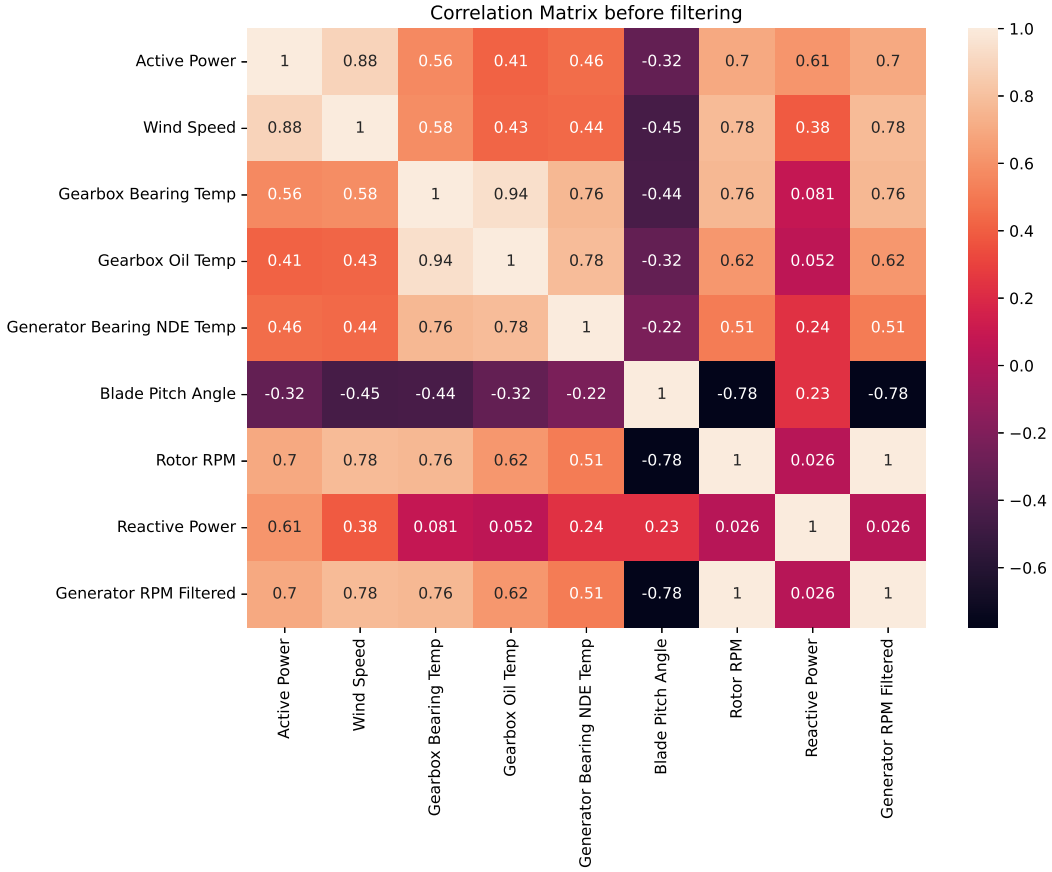


Figure 4.2: Correlation matrix of the training WTGs before filtering operations.

To address these problems, strongly linked characteristics may be eliminated or merged into a single feature. This may simplify the model, enhance its interpretability, and decrease the likelihood of overfitting. By incorporating the correlation between features, it is possible to develop a more accurate and dependable anomaly detection model that is less susceptible to overfitting and other problems.

After all the filtering process correlation matrix can be confronted in order to verify (before and after the process in Figure 4.2 and 4.3) that the removed outlier are characterized by more correlated data, producing a reduction in the overall values of the correlation matrix. From the correlation matrices, an extreme correlation on the magnitudes *Rotor RPM* and *Generator RPM* emerges. In light of this large correlation only rotor RPM will be considered in this, and subsequent experiments.

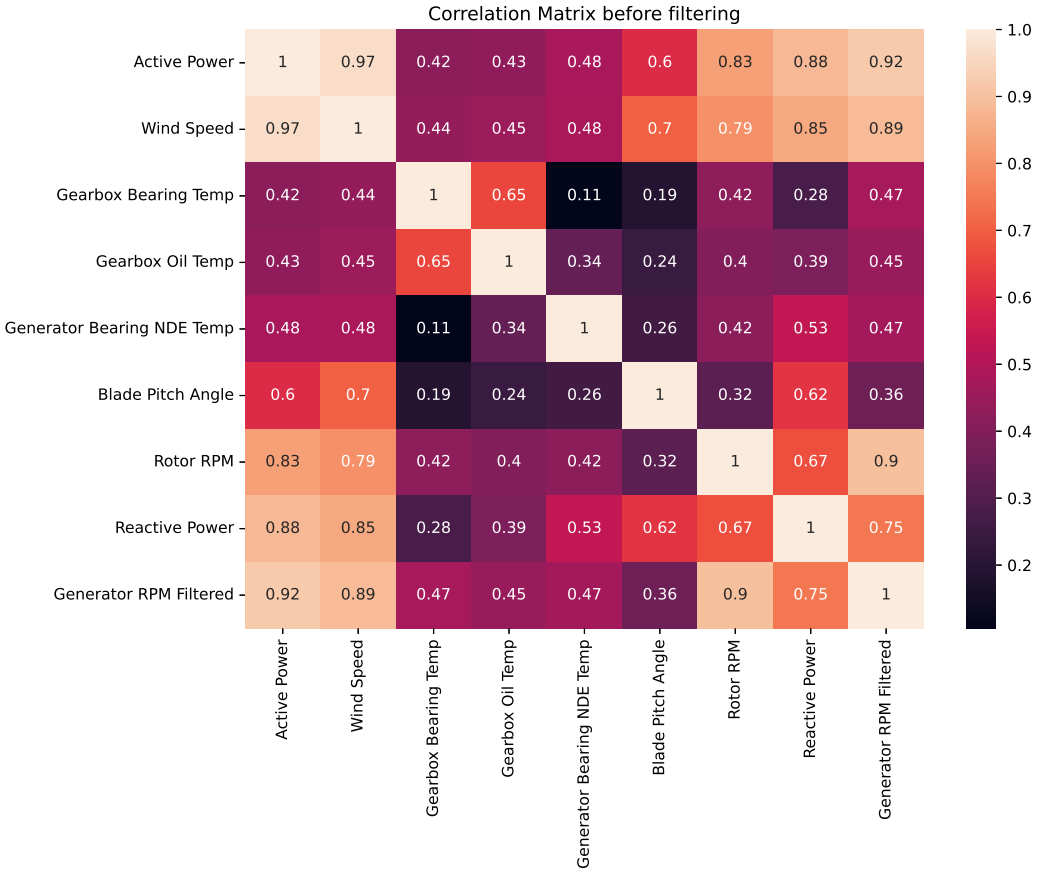


Figure 4.3: Correlation matrix of the training WTGs after filtering operations.

In contrast, *Active Power* and *Wind Speed* measurements after the outlier removal process became more correlated. This phenomenon is due to the fact that by operating with limitations on the power curve both using *cut-in* and *cut-out* speeds and operating on *performance*, cleaner data were obtained around the most linear part of the power curve. Despite the large level of correlation in the experiment both values are retained because, in turbines with anomalous performance these data are the two most relevant in discriminating anomalies.

4.2.5 Sequence Processing

During the preprocessing phase, the previously selected and filtered data are transformed into data sequences suitable for training. Obtaining data sequences from the filtered data is achieved by a sliding windows technique, in which the size of it takes in 12 timesteps (2 hours of 10 minutes averaged data).
The size of the training dataset can be seen in Table 4.7. Data are considered valid for training only if the entire data sequence is associated with a validity label. A single invalid instant allows it to be discarded.

4.3 Models Processing

In this section the AE training task is described. The reconstruction challenge in anomaly detection using AE is the job of training the model to recreate normal, non-anomalous data as precisely as feasible. After the AE has been trained on this normal data, it can be used to identify anomalies by comparing the reconstructed data to the original input data and calculating the difference.

WTG	N° Samples (Seqs.)	% Samples (Seqs.)	% Valid Timesteps
WTG-1	4097/39304	10.42%	25.91%
WTG-2	9223/39304	23.47%	36.15%
WTG-3	8745/39304	22.25%	35.73%
WTG-4	6962/39304	17.71%	37.49%

Table 4.7: Data samples (in sequence format) after filtering and preprocessing. The fact that the number of valid sequences does not equal the number of valid timestamps resides in the fact that only sequences solely consisting of valid data are usable in the training phase.

4.3.1 Reconstruction and Denoise Tasks

In a reconstruction task, the input and target data are the same. The goal of the model is to learn a compressed representation of the input data that can be used to accurately reconstruct the original input data. During training, the input data is fed into the encoder, which generates a compressed representation of the data. This compressed representation is then fed into the decoder, which reconstructs the original input data. The reconstruction error between the input data and the reconstructed output data is used as the loss function during training, and the weights of the autoencoder are adjusted to minimize this reconstruction error.

In a denoising task, the input and target data are not necessarily the same. A denoising AE is a type of model that is designed to learn a representation of the input data that is robust to noise. During training, the input data is corrupted with noise, and the AE is trained to reconstruct the original, noise-free input data. In this instance, the input data and the destination data are not same. The input data are the corrupted version of the original, noise-free data, while the output data are the original, noise-free data. The objective of a DAE is to discover a mapping between noisy input data and noise-free destination data.
Just like a normal reconstruction task, after been trained on normal, non-anomalous data, the DAE may be used to identify anomalies by comparing the difference between the reconstructed data and the original input data. When the reconstruction error between the input data and the reconstructed output data exceeds a predetermined threshold, anomalies will be discovered.

4.3.2 Train and Validation Datasets

In the model training phase, reconstruction metrics will be measured using a validation set. The purpose of this is to verify that the models do not go into overfitting, by learning representation of the training data too weak in generalization. The validation set permits reconstruction to be assessed even on unseen data; hence, it must always be aligned with the training values.

If the validation set is utilized for the denoising process, its significance increases. In this challenge, only the training data are impacted by the injected noise, therefore the model's performance may be evaluated even on undamaged data (such as those from the test phase). The results on the validation set should be better since the time series are easier to rebuild when noise-free, resulting in a smaller reconstruction error.

To properly use a validation dataset, it is necessary to choose a dataset with certain features. The validation dataset must first be indicative of the data that the model is anticipated to meet in the actual world. This indicates that the distribution of classes and features in the validation dataset should be comparable to the distribution in the training dataset. In addition, the validation dataset must be big enough to offer a reasonable assessment of the performance of the model. The validation dataset should be at least 10% of the size of the training dataset, as a general rule. In addition, it is essential to guarantee that the validation dataset is independent from the training dataset, meaning that validation dataset data should not be utilized to train the model.

The whole process is shown in Figure 4.4.

Balance is another crucial attribute of a quality validation dataset. If the training dataset is unbalanced, it is crucial that the validation dataset include an equal number of samples for each

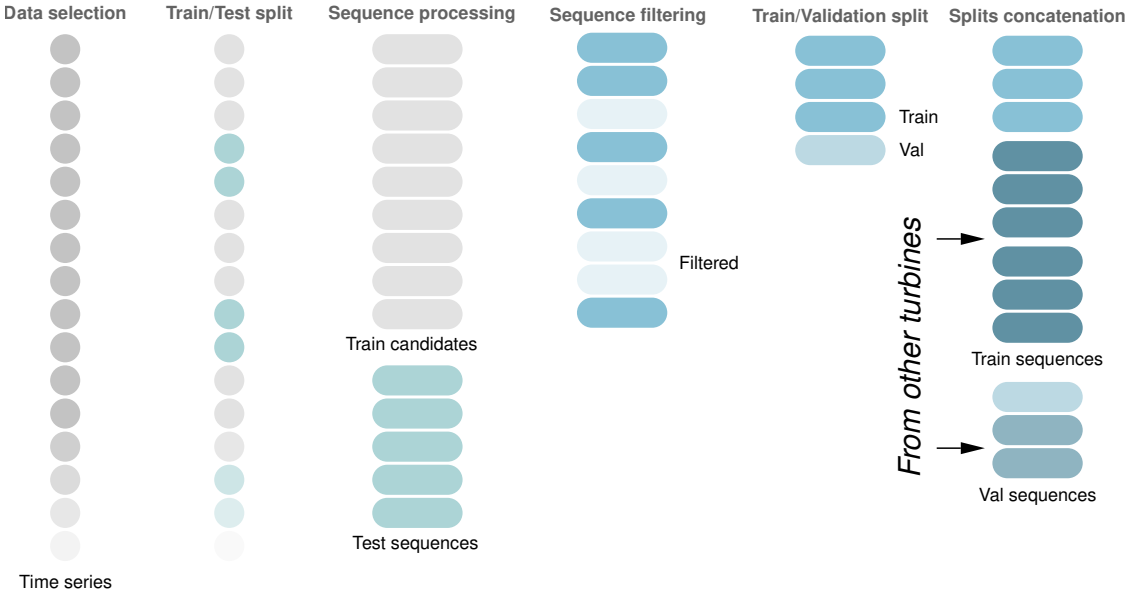


Figure 4.4: Phases of creation of training, validation, and test data sets.

class. The validation dataset should be sufficiently difficult to evaluate the model's capacity to generalize to new, untested data. This implies that the validation dataset should include distinct instances from the training dataset.

4.3.3 Autoencoders Models

This section discusses the architectures of the different AEs involved. The structure of the models used will consist of symmetric encoder/decoder structure, with 2-layers each in the models.

The model are used with a structural difference related to the size of the latent space. Two structures will therefore be used:

- **48 units latent space**: Providing a latent space with 1/2 of the total input dimension.
- **64 units latent space**: Providing a latent space with 2/3 of the total input dimension.

Feed Forward Autoencoders

The first model considered is a feedforward type autoencoder consisting solely of *dense* layers.

In Figure 4.5 it can be observed the initial structure of the AE with latent space with dimension 48.

Figure 4.5: FNN AE initial configuration with latent code dimension of h.

This initial model represents only one of several configurations used for this type of network. Instead, in Figure 4.6 can see the complete model with all the different regularization layers.

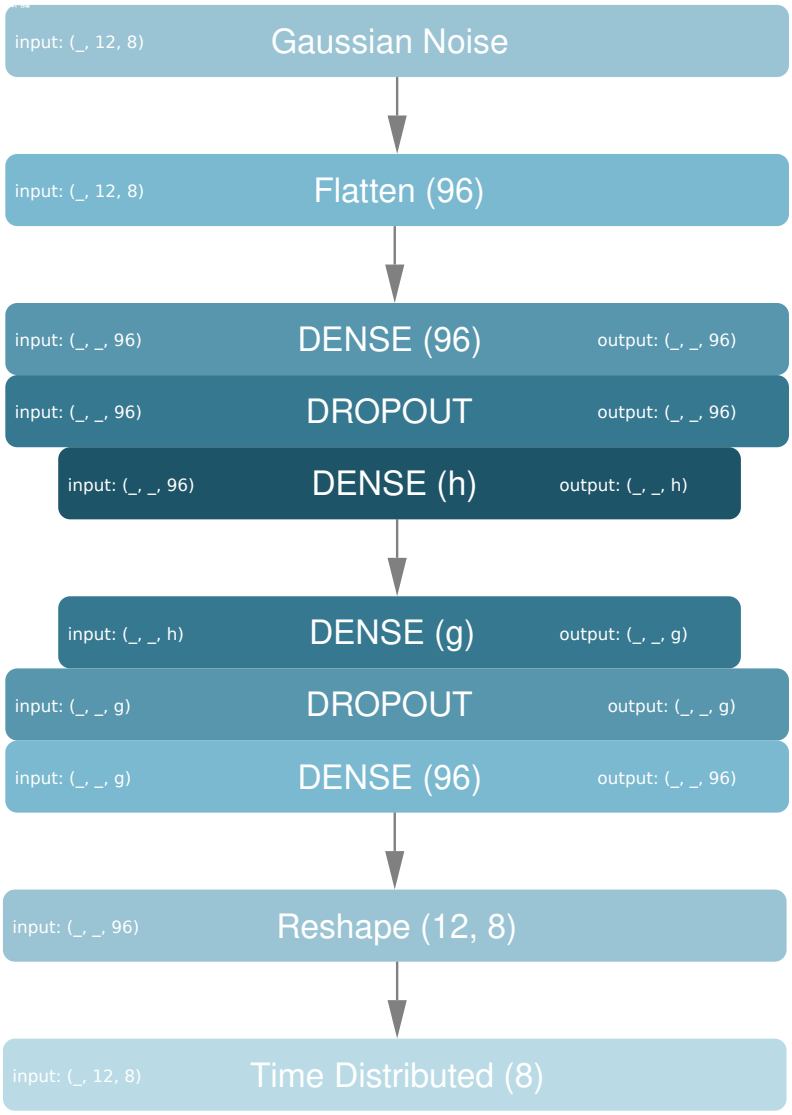


Figure 4.6: FNN AE regularized configuration with latent code dimension of h.

Different hyperparameter configurations will be tested during the training phase and the subsequent validation phase. These configurations make it possible to greatly increase the available combinations and thus the models considered.
The hyperparameters will be:

- **Batch size**: 6, 12, 24 and 48.
- **Dropout rate**: 0.0 and 0.1.
- **Gaussian Noise Variance**: None, 0.05, 0.1 and 0.2.
- **Latent space dimension**: 24, 32 and 48.

Instead, all of the models considered will have the following features in common, the result of experimentation preparatory to that discussed in this thesis.
The fixed parameters are:

- **Activation Functions**: $ReLU$ for the dense layers.

- **Learning rate at start**: 0.01.
- **Loss Function**: MSE.
- **Optimizer**: Adam.

Recurrent Autoencoders

The LSTM and GRU are the recurrent cells used considered in the models. As in the previous scenario, the overall structure of the network consists of an encoder and a decoder, each having two layers. The layers surrounding the bottleneck interact with the different states that the cells are in, while the input and output layers are responsible for accepting and returning the sequence that was formed by the recurrent cells. To be more specific, in order to achieve a high degree of dimensionality reduction, the second layer of the encoder will have the most recent state of the cells as its output, and the first layer of the decoder will interact with the states as its input in order to produce sequences once more. It is typically beneficial to return the hidden state at the final timestep rather than the full sequence since this enables the model to capture long-term dependencies in the input sequence.

The network's middle layer is responsible for ensuring that the states generated by the bottleneck can be reliably replicated along a time sequence. The layer is necessary when use the LSTM or GRU is required to process fixed-length inputs as if they were sequences. It does this by repeating the input a fixed number of times to create a "pseudo-sequence" that can be processed by the LSTM or GRU layer.

In Figure 4.7 is visible the first version of the RNN network in the version with reduced latent space.

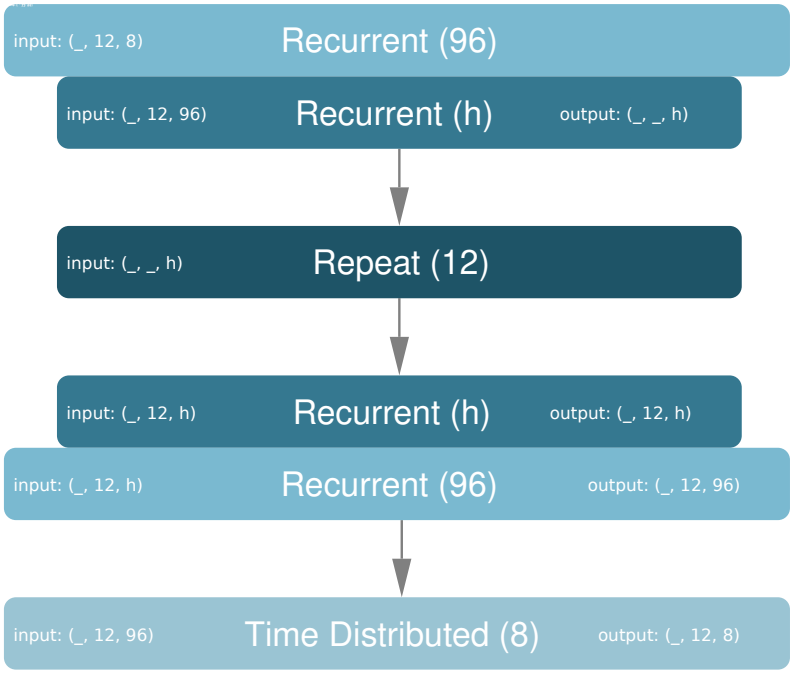


Figure 4.7: LSTM-AE and GRU-AE initial configuration with latent code dimension of h.

The last layer in the network, which appear as a dense layer, can be referred as a *Time Distributed* layer. This layer is a mechanism for applying a dense layer, or any other kind of layer, to each timestep of the output sequence that is created by the LSTM decoder in an AE that uses LSTM or GRU. In an AE that makes use of LSTM, the LSTM decoder is the component that, given the latent representation that was produced by the encoder, is responsible for generating a sequence of output values that should correspond to the input sequence. It is possible to apply

it to each timestep of the output sequence in order to translate the output values to the same dimensionality as the input values.

The *Time Distributed* layer function is to offer a versatile approach of applying a thick layer to each timestep in the output sequence. In particular, it enables the model to acquire distinct weights for each timestep, despite the fact that it continues to make use of the same thick layer throughout all timesteps. In other words, this layer at the end of the network gives the model the ability to recognize temporal patterns in the output sequence and to learn how to map these patterns to the input sequence that corresponds to them.

In Figure 4.8 the regularized configuration of the RNN network is shown.

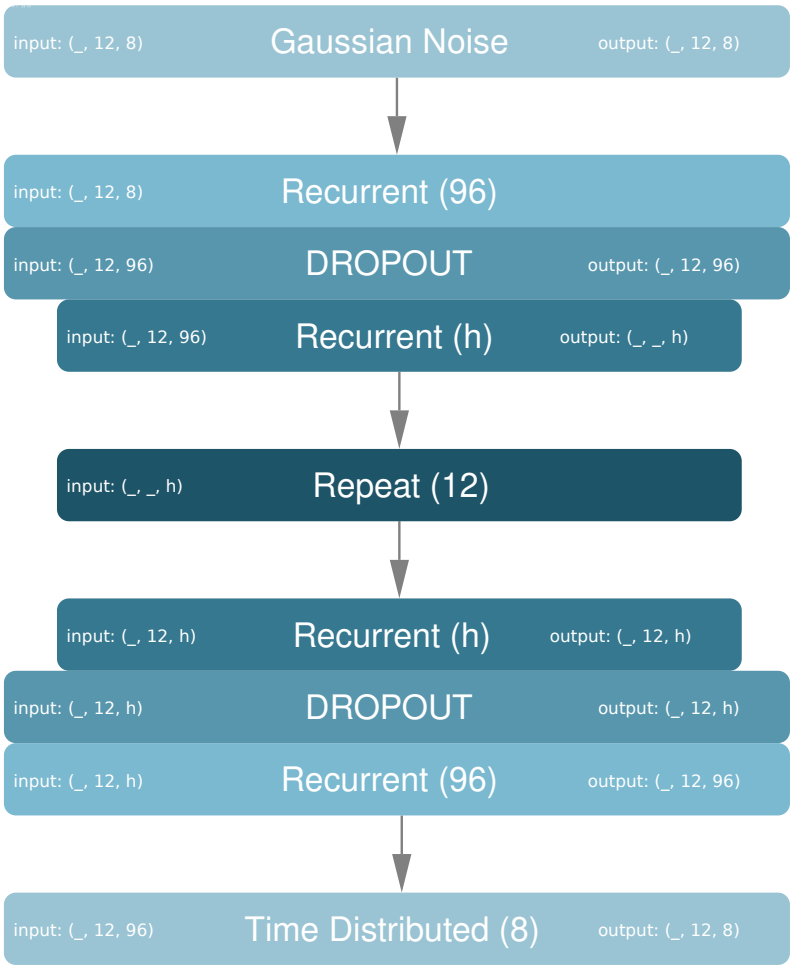


Figure 4.8: LSTM-AE and GRU-AE regularized configuration with latent code dimension of h.

The hyperparameters conisidered are:

- **Batch size**: 6, 12, 24 and 48.
- **Dropout rate**: 0.0 and 0.1.
- **Gaussian Noise Variance**: None, 0.05, 0.1 and 0.2.
- **Latent space dimension**: 48 (1/2) and 64 (2/3).

The unchanged parameters during the process are:

- **Learning rate at start**: 0.01.
- **Loss Function**: *MSE*.

- **Optimizer**: *Adam*.
- **Activation Functions**: *Tanh* for the RNN layers.

4.3.4 Training and Validation Losses

FNN, LSTM and GRU Autoencoders

During training, the convergence speed of a neural network can be affected by a number of variables, including the model's complexity, the quantity and quality of the training dataset, the choice of hyperparameters, and the employed optimization technique. Many factors can affect the convergence speed of a fully connected neural network-based autoencoder (FNN-AE) and a recurrent neural network-based autoencoder (RNN-AE), including the number of layers, the number of neurons per layer, and the input data type.

From Figure 4.9 it is clear how the recurrent models are able to obtain lower losses with a much longer convergence time. It is possible that an FNN-AE can converge faster than an RNN-AE for the same reconstruction task due to the following reasons [33]:

- **Feedforward Architecture**: FNN-AE has a feedforward architecture where information flows only in one direction, from the input layer to the output layer. This makes it easier to optimize the model using standard backpropagation algorithms, which calculate the gradients of the loss function with respect to the model parameters. In contrast, RNN-AE has a recurrent architecture where information loops back from the output layer to the input layer, which can cause instability during training and make optimization more challenging.
- **Size of model**: FNN-AE typically has a smaller number of parameters than RNN-AE. A smaller model size can make it easier to train the model, as there are fewer parameters to optimize. Additionally, smaller models can generalize better, which can result in faster convergence and better performance.

Yet, it is possible that the FNN-AE causes a larger loss at the end of training than the RNN-AE. This may occur if the FNN-AE reaches a suboptimal solution as a result of premature stopping or overfitting. In contrast, the RNN-AE may continue to improve throughout training and arrive at a superior solution, resulting in a reduced final reconstruction loss.

For the identical reconstruction job, a FNN-AE can converge quicker than an RNN-AE because to its feedforward design and reduced model size. However, the FNN-AE may result in a greater loss at the conclusion of training due to overtraining or premature cessation. The design of the autoencoder should be selected depending on the specific job and the properties of the input data, and different architectures should be compared based on their performance on a validation set rather than just the training loss.

Latent Space Dimension

The size of an AE latent space can have a substantial effect on the results of anomaly detection. Each point in the latent space corresponds to a potential representation of the input data. The latent space represents a compressed version of the input data. The difference between the input data and the rebuilt data can be utilized to detect anomalies when an AE is trained to reconstruct the input data from this compressed form.

A wider latent space permits more expressive representations of the input data, which can be useful for spotting abnormalities that are more complex or subtle. Nevertheless, a bigger latent space may also increase the danger of overfitting, which occurs when the AE learns to reconstruct the training data too well and becomes less capable of recognizing anomalies in new or unknown data. A smaller latent space, on the other hand, may be less expressive but more resilient to noise and outliers. This can be useful for finding more evident anomalies, but it may overlook subtler or more sophisticated ones.

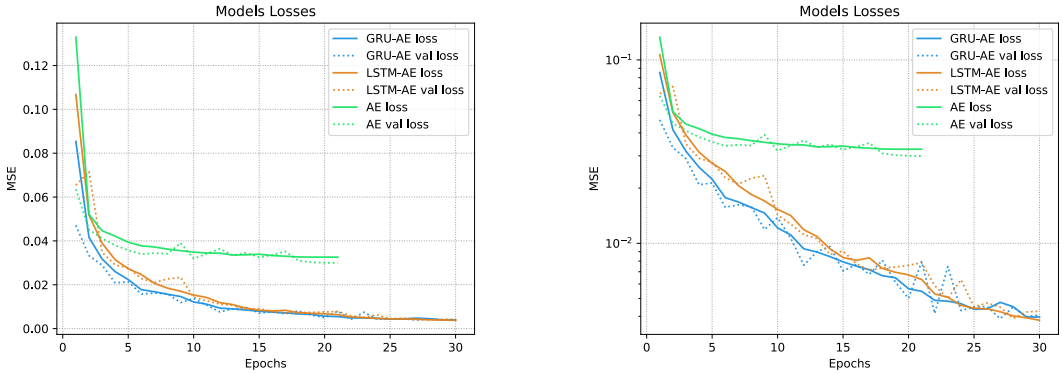


Figure 4.9: Losses from FNN and RNNs AE. Dense AE has few epochs since it models a slightly simpler problem, converging faster. Left: linear scale. Right: Logarithmic scale. Solid line refers to training loss and dotted line refers to validation loss.

Thus, the selection of the latent space size in an AE depends on the unique anomaly detection problem and the trade-off between the expressiveness of the representation and the danger of overfitting. In practice, it may be essential to experiment with various latent space sizes and compare the results of anomaly detection to discover the ideal size for a given problem.

In the problem considered, models with latent space dimensionality ratios of 1/2 and 1/3 to the input data will be used.

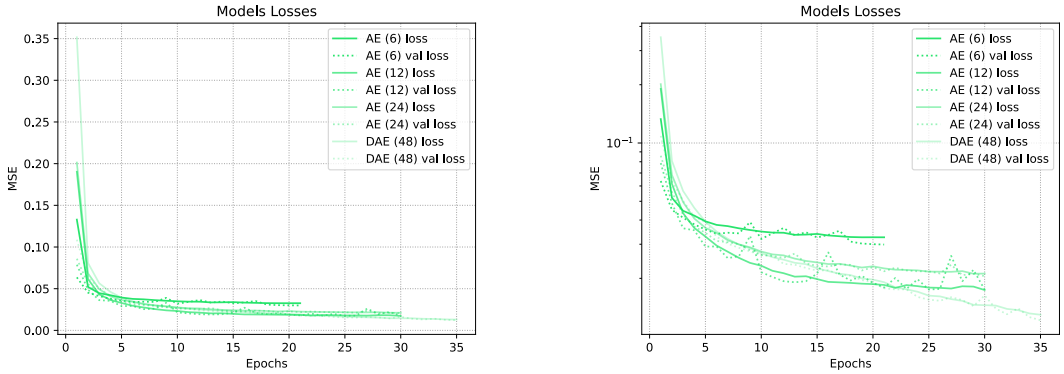
This can be considered a moderate dimensionality reduction, but it is highly limited by the number of available dimension in the starting problem. When the original measures are highly correlated or similar to each other, it may be possible to achieve a higher level of dimensionality reduction without significantly impacting the information content of the data. In contrast, when the original measures are distinct or less correlated, aggressive dimensionality reduction may result in the loss of important information. In this specific case the number of measure is very limited and some of these are unique measures representing a specific sub-component of the WTG. With larger datasets the intensity of dimensionality reduction could have been considerably greater.

Batch Size

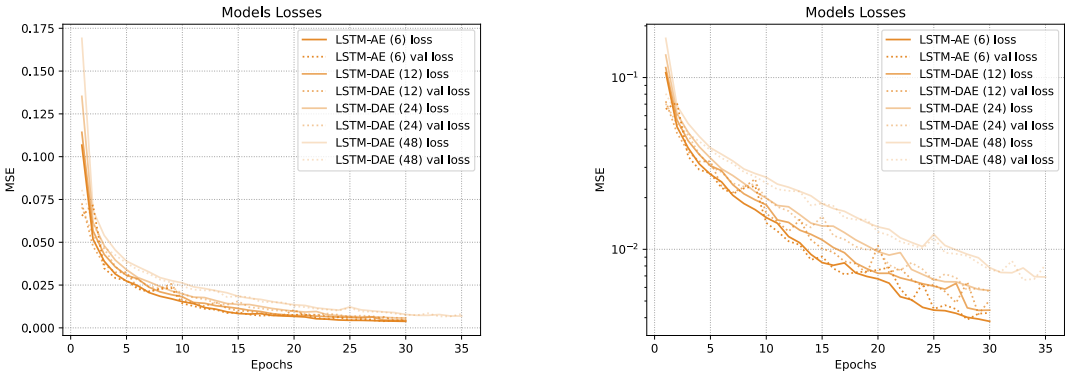
The batch size can have a substantial effect on the training losses of a neural network model.

These are a few examples of how model losses might vary based on batch size:

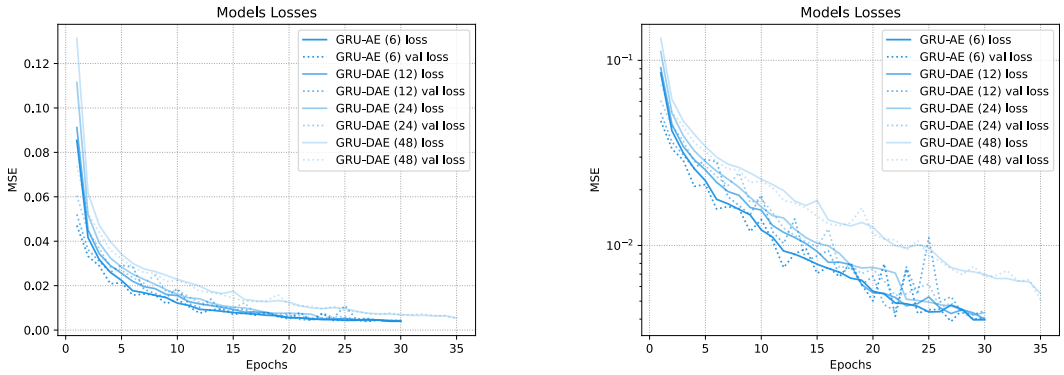
- The gradient estimates used to update the **model parameters are noisier** when the batch size is small because they are based on a smaller sample of the training data. This can result in more unstable training and slower convergence, leading to greater loss values.
- Lower batch sizes might also result in **less precise optimization** since the optimizer may be less able to locate the global minimum of the loss function. This can lead to increased ultimate loss values and subpar model performance.
- Small batch sizes can also have a **regularization effect** because the model is exposed to more variability in the training data. This can aid in preventing overfitting and enhancing generalization performance, resulting in reduced loss values on the validation set.
- A larger batch size means that fewer updates are made to the model parameters during each training iteration. This can result in **slower training**, as it takes longer to process each batch of data.
- If the batch size is too large, it can also result in **poor generalization performance**. This is because the model is less likely to encounter rare or unusual examples during training, which can make it less able to handle such cases during inference.



((a)) Losses from FNN-AE with different batch size.



((b)) Losses from LSTM-AE with different batch size.



((c)) Losses from GRU-AE with different batch size.

Figure 4.10: FNN and RNN AE, models losses at different batch sizes. The parameter after the model name refers to batch size. Left: linear scale. Right: Logarithmic scale. Solid line refers to training loss and dotted line refers to validation loss.

In general, the best batch size for a given model will depend on the unique task and architecture, as well as other considerations such the available computational resources and the amount of the training dataset. In reality, it is frequently advantageous to experiment with various batch sizes during model training and select the one that yields the best results on the validation set.

From the results in Figure 4.11, it can be easily observed that batch size has mixed effects among different AEs architectures. In particular in recurrent models the use of very large batch sizes (48-sample size) produces large losses and extremely long convergence times. The overall result in these models makes it possible to say with great confidence that the effect of batch size is easily explained by the reduction of optimization steps in the models across epochs. However, the

effects on the FNN model gives unsuspected results, leading it to obtain better scores the more the batches increase in size. Probably due to the fact that the regularization effect introduced from the small batches resulted too pronounced on a much simpler model than on the more complex RNNs.

Different batch sizes will be tested in the experimental phase: starting with 6 sequences at a time, sizes at 12, 24 and 48 will be tried. However, only the approaches with size at 6 and 12 will be further investigated with other parameters, while the remainder will have purely demonstrative value in this training phase.

Denoising Task Noise Intensity

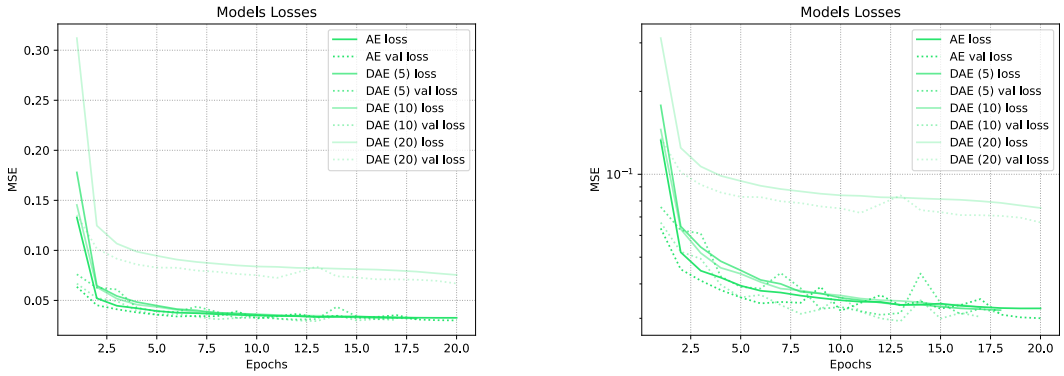
In AE, the model is trained to minimize the reconstruction error between the input and the output. As there is no additional noise removal step, the model learns to encode the input data as accurately as possible.

However, in DAE, the model is trained to remove the added noise and reconstruct the original input. Therefore, the DAE has an additional objective to remove noise from the input and reconstruct the original signal, which can be more challenging than the simple reconstruction task in AE.

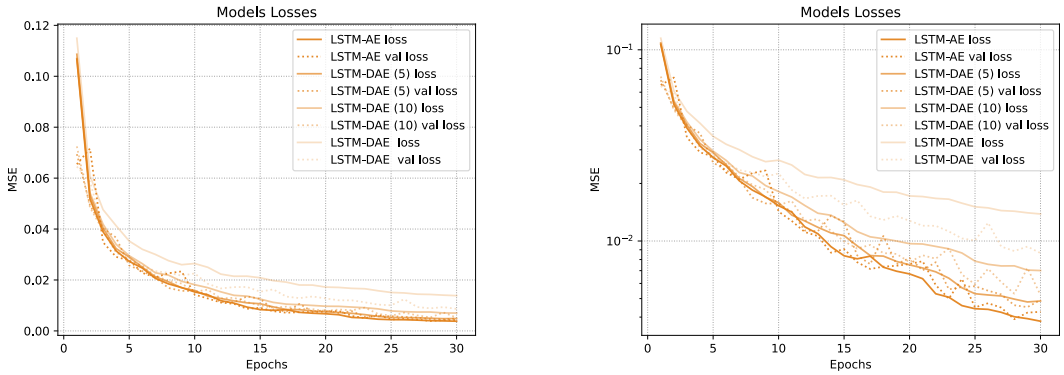
As the noise variance increases in DAE (Figure 4.10), the problem becomes harder as the model needs to remove more severe noise from the input signal. In contrast, in AE, the model only needs to reconstruct the input signal, and the added noise does not affect the reconstruction loss significantly. Therefore, in general, the reconstruction problem in AE has a lower loss than the DAE problem, especially at higher noise variances. For this reason reduced loss value is found in all model variations considered.

The effect of higher error occurs not only on the training data but also on the validation data, although they are not directly altered by the effect of noise. The reason for this could be in general a more difficult tendency to learn the reconstruction task for denoise-type models.

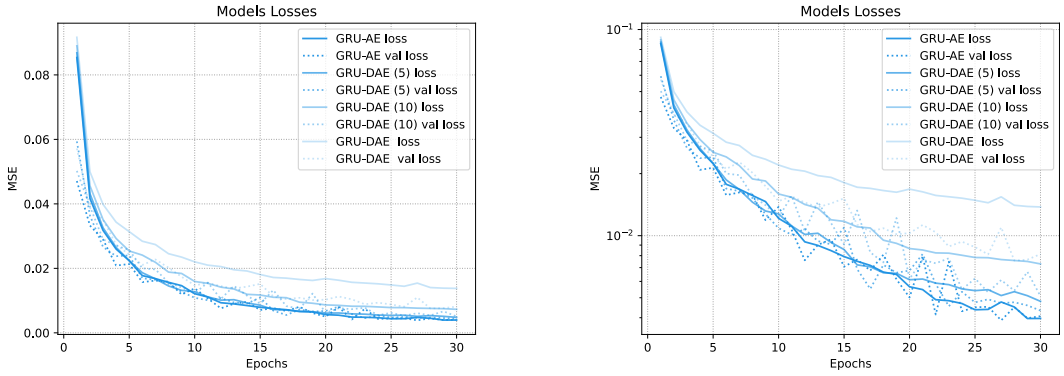
In the process of denoising, the models will employ as a Gaussian noise parameter, the value of a relative variance in relation to the data that is input into the model. The values vary from 5 percent, to 10 percent, and then finally all the way up to 20 percent of the variance in the training data. Taking into account the fact that each of the training data has a unit variance, the mistake that is introduced will have a variance that is equal to the decimal value that is taken into account. Due to the fact that the mistakes will have averages that are not zero, the lags that are brought about by the Gaussian disturbance will be both negative and positive.



((a)) Losses from FNN-AE with different noise power.



((b)) Losses from LSTM-AE with different noise power.

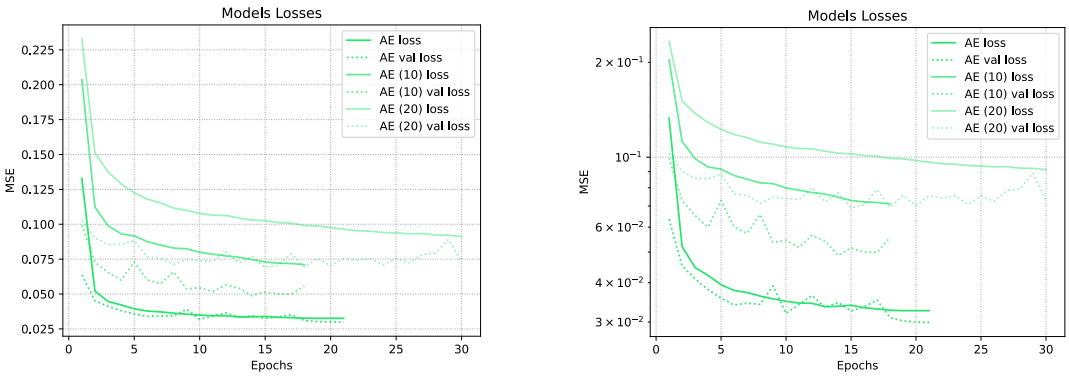


((c)) Losses from GRU-AE with different noise power.

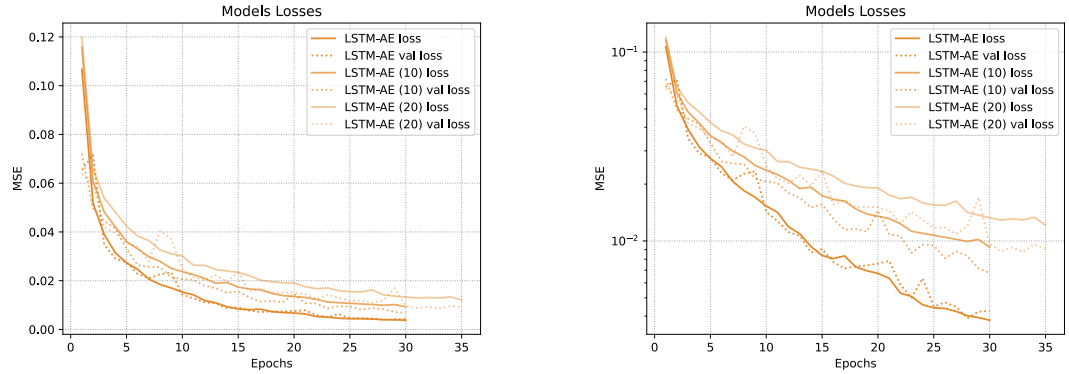
Figure 4.11: DAE (and AE, since it has no noise) models losses at different noise power. The parameter after the model name refers to the variance of the noise added in the input data (data is monovariate after standard scaling technique). Left: linear scale. Right: Logarithmic scale. Solid line refers to training loss and dotted line refers to validation loss.

Dropout

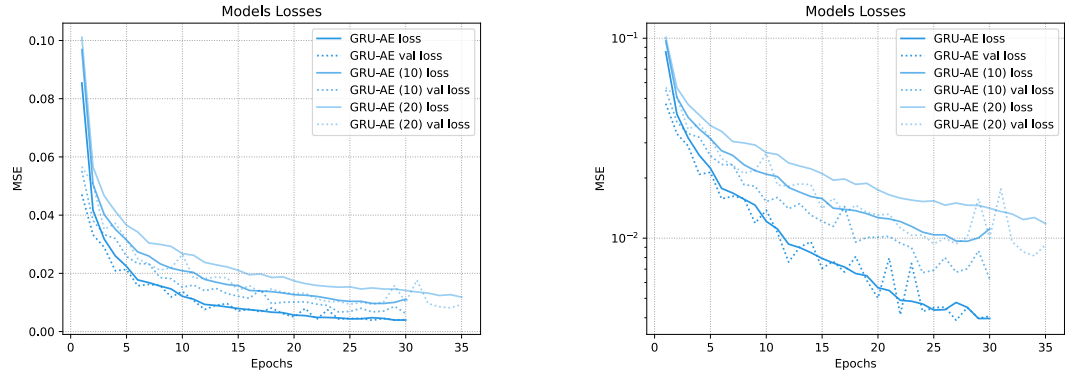
A percentage of neurons in a layer are chosen at random to be ignored after each forward training pass when dropout is applied. This reduces the reliance between the neurons, which can enhance the network's capacity for generalization.



((a)) Losses from FNN-AE with different dropout rate.



((b)) Losses from LSTM-AE with different dropout rate.



((c)) Losses from GRU-AE with different dropout rate.

Figure 4.12: AEs models losses at different dropout rate. The parameters refer to the percentage of neuron deactivated during each training step. Left: linear scale. Right: Logarithmic scale. Solid line refers to training loss and dotted line refers to validation loss.

The effect of dropout on the loss of a neural network model might vary based on the design and data used, but in general, dropout will reduce the loss performances of the model during training. This is because dropout prevents overfitting and promotes the network to discover more robust and generalizable characteristics.

It is also important to note that attrition can make training more difficult and hinder convergence. This is due to the employment of a distinct group of neurons on each forward pass, which might result in more unstable gradients and slower convergence. Thus, it is typical to combine dropout with other regularization approaches, such as weight decay, to further enhance the network's performance.
It is evident from Figure 4.12 how the dropout acted by reducing the ability of the model to converge rapidly is had in effect of raising the loss is thus reduced the overfitting This technique therefore has effects that are the same as the AE denoise case.

Throughout the course of the experiment, different dropout rates will be employed. The percentage of artifact neurons that remain inactive after a forward pass constitutes the dropout benchmark. In the particular example of the experiment, the levels that will be used are going to be 10% and 20%. While better levels of regularization will be achieved with the use of other approaches, additional dropout benchmarks will not be considered at this time.

Graphs of losses using mixed regularization techniques will not be reported, but instead results on anomaly detection tasks related to them will be reported.

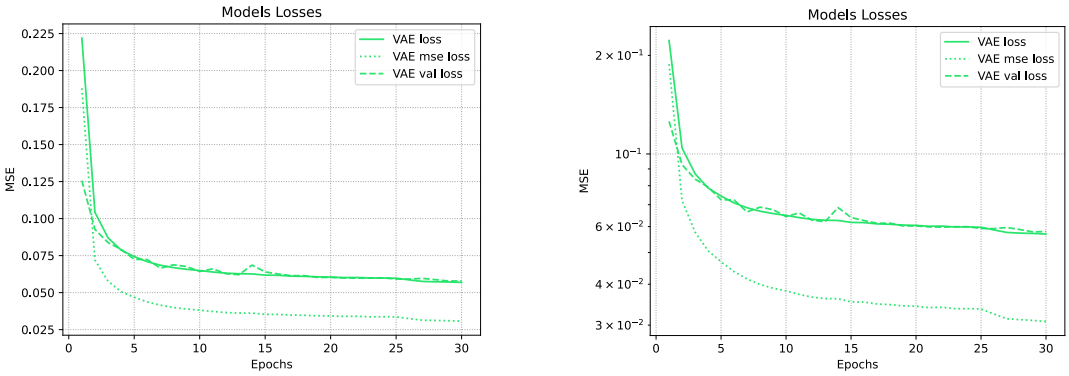
VAE

VAEs are potent generative models commonly employed for unsupervised learning and data generation. Nonetheless, training VAEs can be problematic due to the complexity of maximizing the trade-off between reconstruction loss and KL divergence.

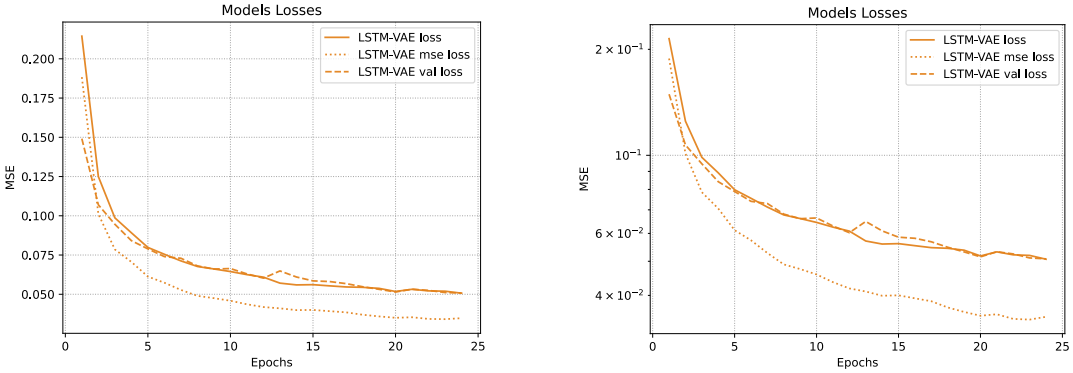
The reconstruction loss represents the difference between the input data and the output of the decoder network, whereas the KL divergence measures the difference between the latent variable distribution and a previous distribution (usually a standard normal distribution).

The beta parameter β, which adjusts the KL divergence term in the loss function, controls the trade-off between the two terms. A high value of β reduces the variance in the latent space, but can cause the model to be inadequately fitted to the data. A low value, on the other hand, results in a greater variation in the latent space, but can cause the model to overfit the data. Consequently, determining the ideal parameter value is essential for training a VAE.

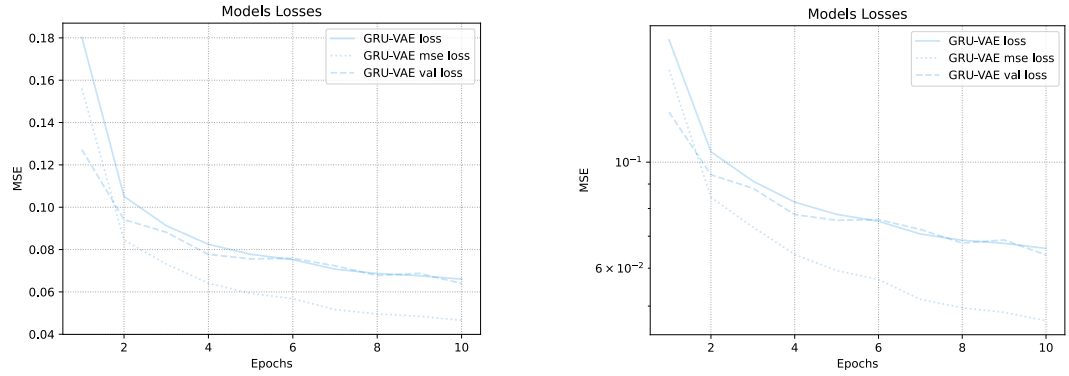
In conclusion, it might be challenging to train a VAE due to the delicate trade-off between the reconstruction loss and the KL divergence term. The β parameter plays a critical role in achieving a balance between these two variables, and its optimal value must be determined through thorough experimentation. Losses can be seen in Figure



((a)) Losses from FNN VAE.



((b)) Losses from LSTM-VAE.



((c)) Losses from GRU-VAE.

Figure 4.13: VAEs models losses (β at 0.01). The parameters refer to the percentage of neuron deactivated during each training step. Left: linear scale. Right: Logarithmic scale. Solid line refers to training loss and dotted line refers to validation loss.

4.3.5 Residual Evaluation

At the end of the training process, the models can be evaluated in their reconstruction capability using the evaluation of their residuals. After the health reference model is trained, the can multivariate residuals be obtained.

From the residual the aggregation metric is computed (MSE in this case) allowing to produce an anomaly indicator. With this indicator all fault can be detected by choosing a threshold which allow from a discrimination of the sample when the value is exceeded.

When monitoring indicators are produced from a sequence using Mean Squared Error (MSE), a distribution can be computed to determine a threshold for anomaly detection. Traditionally, a Gaussian distribution has been used for this purpose, but in some cases, a Kernel Density Estimation (KDE) can provide more accurate results.

Estimating the probability density function of a random variable can be done using the KDE technique, which is a non-parametric approach. Using a kernel function at each data point, it constructs a continuous probability density function, which is then used to make an estimate of the underlying probability density function of a given set of observations. This estimate can then be used. Estimating the likelihood of seeing a particular value can be done with the help of the probability density function that was produced as a result. KDE is able to adjust to the shape of the data and provide a more accurate representation of the distribution that it is based on, in contrast to the Gaussian distribution, which assumes that the shape will always be the same. Because of this, it is a useful tool for anomaly detection because it is able to identify anomalies that may not be captured by a Gaussian distribution. Consequently, this makes it a useful tool.

Formally, KDE is defined as, given a sample population $X = \{x_1, x_2, ..., x_n\}$, the estimated probability density function $\hat{f}(x)$ using the Gaussian kernel is defined as:

$$\hat{f}(x) = \frac{1}{nh\sqrt{2\pi}} \sum_{i=1}^{n} e^{-\frac{(x-x_i)^2}{2h^2}}$$

where h is the bandwidth parameter, which controls the smoothness of the estimate, and is typically selected using a method such as cross-validation.

In this work the Gaussian kernel $K(u)$ is used, is defined as:

$$K(u) = \frac{1}{\sqrt{2\pi}} e^{-\frac{u^2}{2}}$$

where u is the distance from the observation to the point at which the density estimate is evaluated.

- x is the value at which the density estimate is evaluated.
- K is the kernel function, which determines the shape of the density estimate.
- h is the bandwidth, which controls the smoothness of the density estimate. A smaller bandwidth produces a smoother estimate but may oversmooth the data, while a larger bandwidth may capture more detail but may be more sensitive to noise.

It does this by computing a weighted sum of kernel functions centered at each observation, with the weight of each kernel determined by its distance from x and scaled by the bandwidth h. The resulting density estimate represents the probability of observing a value x from the underlying distribution.

In Figure 4.14 GRU-AE distribution and PDF resulting from KDE are shown.

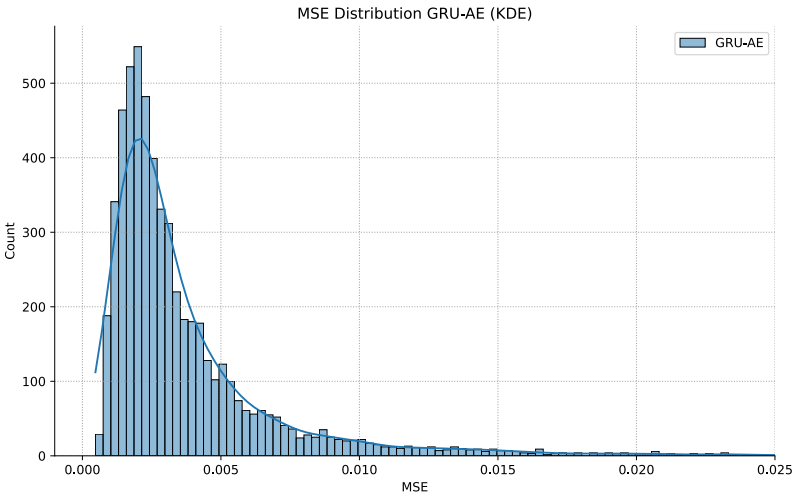


Figure 4.14: MSE distribution and PDF from KDE estimation on validation set.

In Figure 4.15, residual are shown for few models involved in training.

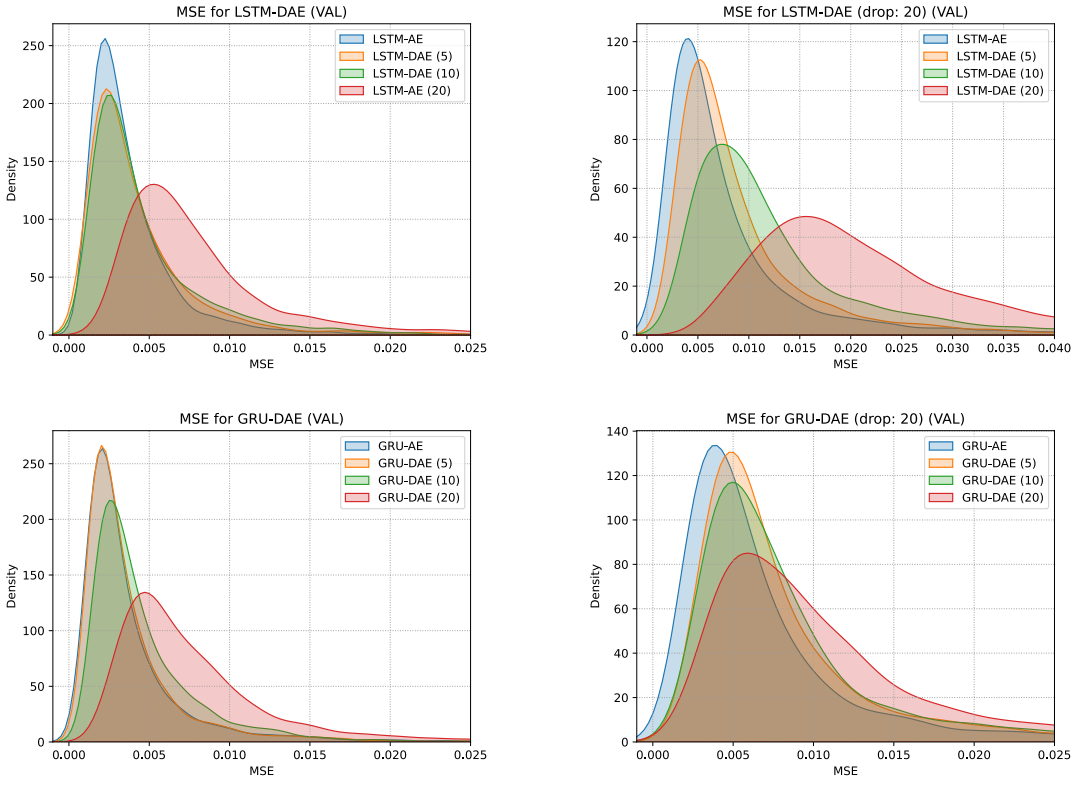


Figure 4.15: Residual PDF distribution using KDE on several models residual over the validation data for reconstruction and denoise tasks.

4.4 Anomaly Detection in Validation and Testing Datasets

In multivariate anomaly detection, reconstruction error is generally employed as a measure of how abnormal a data point is. This is because reconstruction error takes into account multiple variables simultaneously. After the calculation of the reconstruction error for each data point in our dataset, labels are aligned to the data points and evaluate the performance of our model by plotting ROC curves.

Here are the general steps to go from reconstruction error to labeling and ROC evaluation:

- Determine a threshold value for the reconstruction error. This threshold value will be used to classify data points as normal or anomalous. One way to determine the threshold is to use a validation set and choose a threshold that balances the false positive rate (FPR) and true positive rate (TPR).
- Classify each data point as normal or anomalous based on whether its reconstruction error is above or below the threshold value.
- Compute the confusion matrix based on the true labels and predicted labels. The confusion matrix will have four elements: true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN).
- Calculate the TPR and FPR for different threshold values by varying the threshold value and computing the corresponding TPR and FPR. These values can be plotted on a ROC curve.
- Evaluate the performance of the model by computing the area under the ROC curve (AUC). A higher AUC indicates better performance.

It's important to note that the choice of threshold value will affect the labeling of data points and the resulting ROC curve. Therefore, it's important to choose an appropriate threshold value that balances the trade-off between the FPR and TPR.

In symbols, let X be the original multivariate time series, and let X' be the reconstructed time series using a reconstruction algorithm such as PCA or autoencoder. Let e(t) be the reconstruction error at timestep t, which is defined as the difference between X(t) and X'(t):

$$e(t) = X(t) - X'(t)$$

To obtain a measure that can be used for anomaly detection at each timestep, the reconstruction errors is aggregated across all variables using an aggregation function f. This function can be any suitable measure of distance between the original and reconstructed time series. The types of functions are listed later.

Let F be the aggregation function, and let $f(t)$ be the resulting measure at timestep t, which is defined as follows:

$$f(t) = F\big(e(t)_1, e(t)_2, \ldots, e(t)_n\big)$$

where $e(t)_1$, $e(t)_2$, ..., $e(t)_n$ are the reconstruction errors for each variable at timestep t.

For example, with MSE as the aggregation function, $f(t)$ can be defined as follows:

$$f(t) = \frac{1}{n}\sum_{i=1}^{n} e(t)_i^2$$

where n is the number of variables in the multivariate time series.

Once the measure $f(t)$ is calculated for each timestep, it can be compared to a threshold to detect anomalies. If $f(t)$ exceeds the threshold, that timestep can be flagged as anomalous.

The aggregation function used before can be one of the following:

- **Maximum reconstruction error**: In this method, the maximum reconstruction error is taken across all the measures as the anomaly score for that data point. This method assumes that the measure with the highest reconstruction error is the most important in detecting anomalies.
- **Average reconstruction error**: In this method, the average of the reconstruction errors is computed across all the measures as the anomaly score for that data point. This method assumes that all the measures are equally important in detecting anomalies.
- **Mahalanobis distance**: In this method, the Mahalanobis distance is obtained between the reconstructed data point and the mean of the normal data points. The Mahalanobis distance takes into account the covariance of the data and provides a more accurate measure of distance than Euclidean distance.
- **Principal component analysis (PCA)**: In this method, PCA is applied to the reconstruction errors and take the first principal component as the anomaly score. The first principal component captures the most variance in the data and is often a good representation of the overall anomaly score.

Once a single anomaly score is obtained for each data point, it can be used to plot the ROC curve and evaluate the performance of our anomaly detection model. The choice of method for aggregating the reconstruction errors depends on the specific application and the characteristics of the data.

In the case studied by this thesis, the aggregation metric for the data in the anomaly detection stage is the MSE. Therefore, after obtaining the scores in the testing phase of the model, the timestamps for the reconstructed sequences are used to calculate a difference the values obtained, which is then squared and aggregated with an averaging operation.

The anomaly task, in some tests will then be used at different levels of complexity to determine the accuracy of the algorithm:

- **Time series anomaly detection**: In this first task, the objective is determining if the entire time series is anomalous, is commonly known as anomaly detection for time series data.
- **Point anomaly detection**: In this task, each individual timestep of time series will be tested for anomalies.

4.4.1 Event Discrimination in Validation

A Status discrimination is performed as an initial test to verify the efficacy of the models in the anomaly detection phase of the process.

At this point in the process, data that has been labeled using the same labels that were used to discriminate data for training can be used. In particular since the discrimination involves whole sequences, RE is aggregated in the sequence with an MSE (between all measures and timesteps inside the sequence). The labels referring to an anomalous sequence are obtained by aggregating the single timesteps labels using a logical-OR between all the single timesteps labels, plus a performance average between all the timestep of a sequence. The time sequences that contain only one timestep that is reported as being abnormal will be eliminated from the dataset in an effort to cut down on the number of borderline situations.

It is also possible to employ a partial method, in which only particular subsets of labels are to be evaluated. The requirement here is that the subsets in question contain partitions with higher reconstruction error, or that they only use partitions with both positive and negative labels. In this partition test, only few filter conditions are considered.

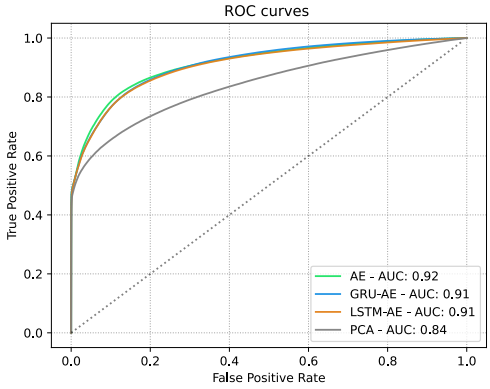


Figure 4.16: ROC curves classifications results with the main models. The same filters used in training are used for labelling the data.

In Figure 4.16, it can be seen that all models are easily able to bring great results in the classification phase by trying to positively label the data according to the splits used in the filters of the training phase. The models in the image are all the basic version of the AEs with 48 latent dimension, with batch 12. The number of principal components for PCA is determined when the cumulative variance contribution rates exceed 80%, 90% and 95%.

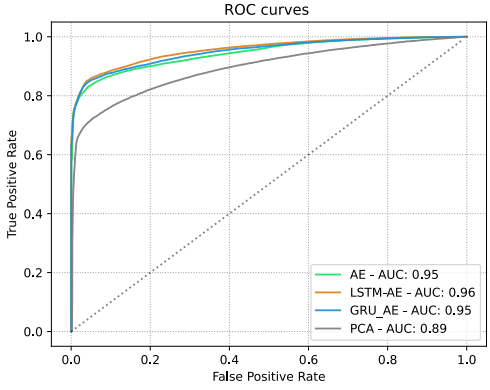
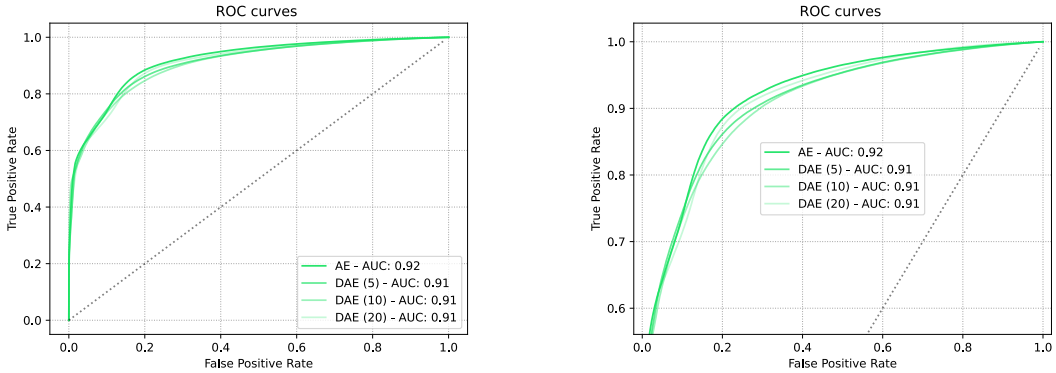
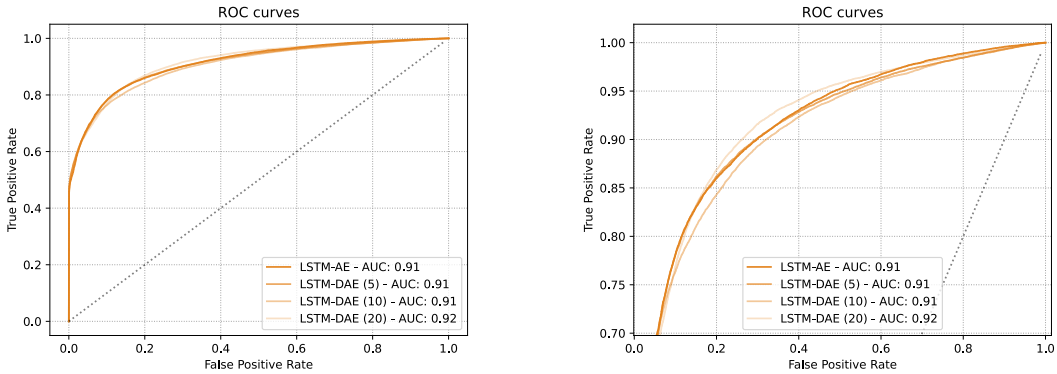


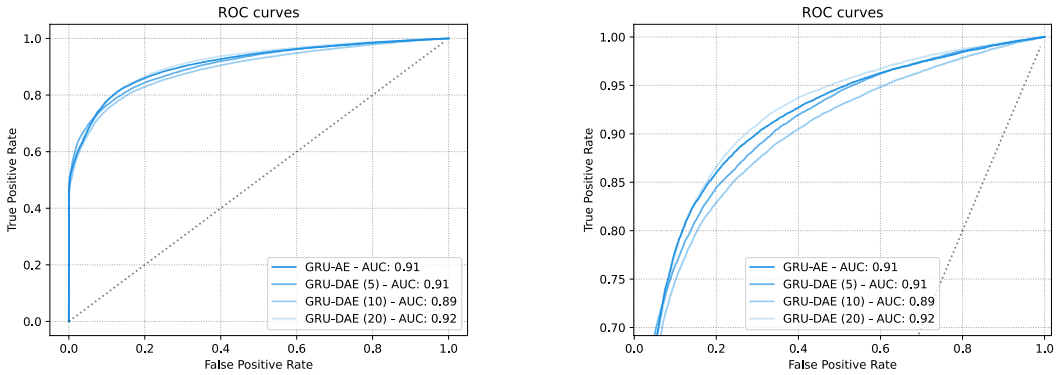
Figure 4.17: ROC curves classifications results with the main models. Labels refers to the *Active Status* vs *Error* and *Inactive* ones.



((a)) ROC curves for AE.



((b)) ROC curves for LSTM.



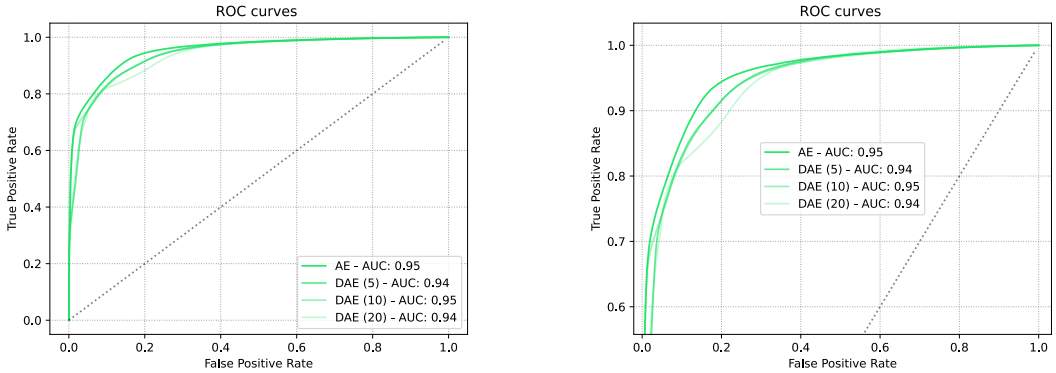
((c)) ROC curves for GRU.

Figure 4.18: ROC curves for all filters involve in training for AE and DAE models. Left: whole curve. Right: curve zoomed on high TPR. Model with dropout at zero and batch size at 6, latent space dimension of 48.

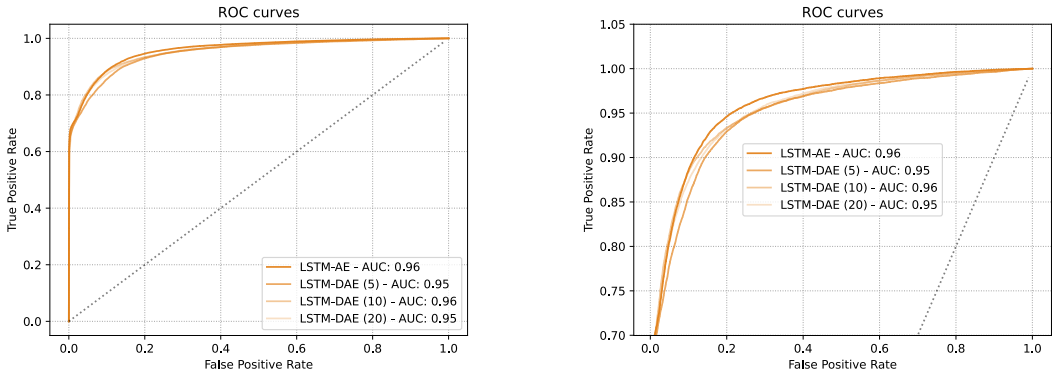
In Figure 4.18, different roc curves are displayed for different DAE models, making case for the highest TPR values. The image shows that all models achieved comparable performance, and the models with more pronounced noise produced the greatest results in the case of RNNs. In Figure 4.17, on the other hand, it is possible to visualize the same patterns in the classification stage using the turbine *Status* in the observed timestamp as the discriminating label.

From Figure 4.19 it can be observed, all DAE models, both FNN and RNN, are able to achieve very high TPR values with very low FPR values. This demonstrates that this type of event is characterized by a high reconstruction error relative to the observed quantities.

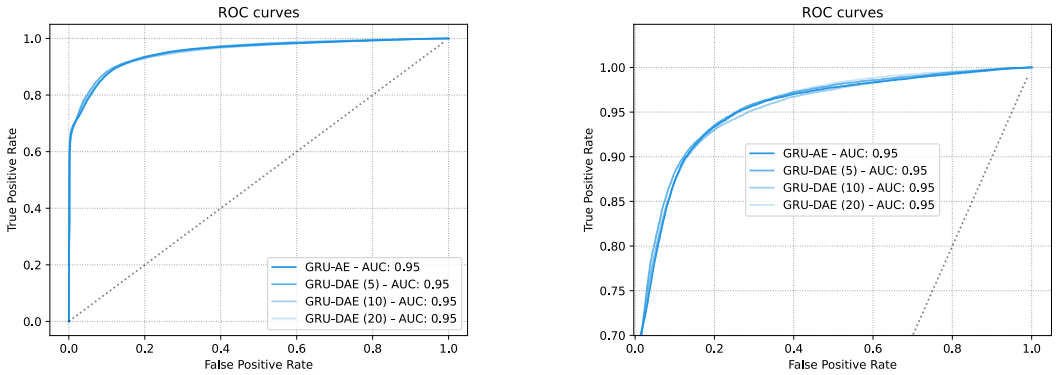
The fact that the results are greater demonstrates that the times when the status becomes



((a)) ROC curves for AE.



((b)) ROC curves for LSTM.



((c)) ROC curves for GRU.

Figure 4.19: ROC curves for *Error* and *Inactive* statuses for AE and DAE models. Left: whole curve. Right: curve zoomed on high TPR. Model with dropout at zero and batch size at 6, latent space dimension of 48.

inactive or error, the reconstruction error obtains the most deviant results from the normality of the data observed in the training phase, and as a result, it is easier to observe such differences than data that are characterized only by lower performance. This is because the reconstruction error obtains the most deviant results from the normality of the data observed in the training phase. In the context of the results, it is not difficult to deduce that the model is endowed with an excellent capacity for detecting the filters that were applied in order to gain access to the training phase. The faults in the reconstruction turn out to be extremely interchangeable on the states that the turbine is in, whether it be error or idle (in Figure 4.20).

In the tables below, all model results are shown at varying batch size, dropout for models with

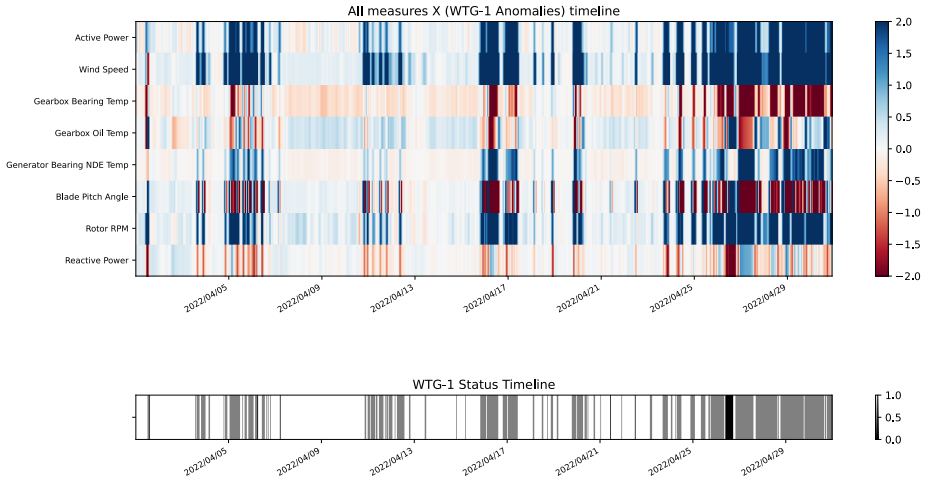


Figure 4.20: Reconstruction error for all the different measures and the *Status* code representing *Error* statuses in black and *Inactive* statuses in gray.

latent space at 32 for FNN (Table 4.8), LSTM (Table 4.9) and GRU (Table 4.10).

A bar graph is shown in the Figure 4.21 where one can see the trend of ROC-AUC score as the hidden space dimension changes in the training filter test. Although the values are within a small range the best results are obtained with hidden dimension at 48. In fact, the experiments show the different results with value of the parameter at 32 and 48 units.

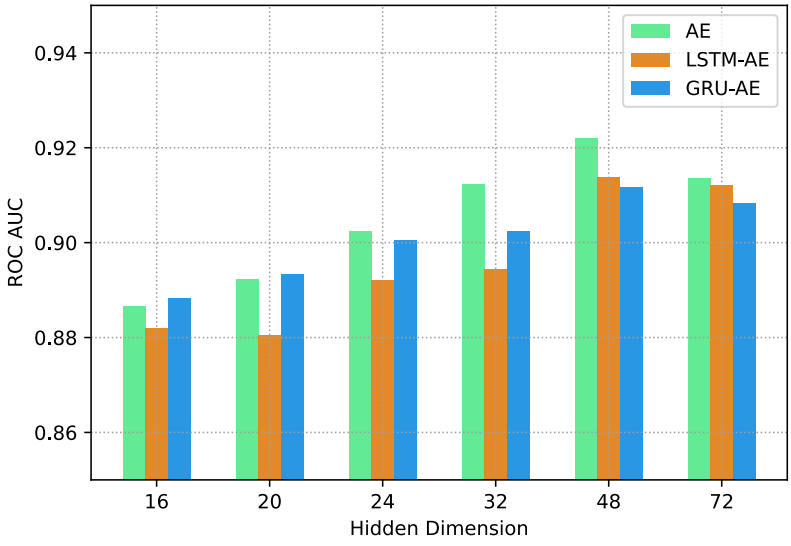


Figure 4.21: ROC-AUC score for the main model with different latent space dimension. These values refers to the the train filters test on validation data, with batch size at 6, 0% dropout rate and no noise. Maximum values are reached in all the models at 48 which is chosen as main parmeter.

		Dropout Rate 0%	10%
Batch-6			
Noise	**NO**	**0.9123**	0.907
	5%	0.912	0.9066
	10%	0.9110	0.9039
	20%	0.9098	0.9002
Batch-12			
Noise	**NO**	**0.9190**	0.9115
	5%	0.912	0.902
	10%	0.9139	0.9054
	20%	0.9120	0.9010

		Dropout Rate 0%	10%
Batch-6			
Noise	**NO**	**0.9511**	0.9425
	5%	0.9452	0.9410
	10%	0.9504	0.9400
	20%	0.9456	0.9393
Batch-12			
Noise	**NO**	**0.9502**	0.9456
	5%	0.9489	0.9432
	10%	0.9467	0.9458
	20%	0.9457	0.9384

Table 4.8: FNN AE (latent dimension of 32) results in terms of ROC-AUC scores. The tables show different results based on combinations of batch-size, noise power and dropout probability in each model. Left: Results for discrimination based on the same filters used in training. Right: Results for discrimination using only *Error* and *Inactive* statuses.

		Dropout Rate 0%	10%
Batch-6			
Noise	**NO**	0.8944	0.8992
	5%	0.9086	0.9142
	10%	0.9104	**0.9192**
	20%	0.9171	0.9181
Batch-12			
Noise	**NO**	0.8925	0.892
	5%	0.9066	0.9035
	10%	0.9129	**0.9187**
	20%	0.9141	0.9181

		Dropout Rate 0%	10%
Batch-6			
Noise	**NO**	0.9529	0.9544
	5%	0.9525	0.9528
	10%	0.9532	**0.9584**
	20%	0.9547	0.9447
Batch-12			
Noise	**NO**	0.955	0.9562
	5%	0.9549	0.9584
	10%	0.9509	**0.9597**
	20%	0.9588	0.9568

Table 4.9: LSTM-AE (latent dimension of 32) results in terms of ROC-AUC scores. The tables show different results based on combinations of batch-size, noise power and dropout probability in each model. Left: Results for discrimination based on the same filters used in training. Right: Results for discrimination using only *Error* and *Inactive* statuses.

		Dropout Rate 0%	10%
Batch-6			
Noise	**NO**	0.9023	**0.9091**
	5%	0.8997	0.9012
	10%	0.8932	0.9016
	20%	0.8973	0.9026
Batch-12			
Noise	**NO**	0.9031	**0.9103**
	5%	0.9003	0.9025
	10%	0.8964	0.9010
	20%	0.8984	0.9005

		Dropout Rate 0%	10%
Batch-6			
Noise	**NO**	0.9547	**0.9562**
	5%	0.9512	0.9524
	10%	0.9459	0.9461
	20%	0.9488	0.9461
Batch-12			
Noise	**NO**	**0.959**	0.9476
	5%	0.9510	0.9476
	10%	0.9494	0.9468
	20%	0.9488	0.945

Table 4.10: GRU-AE (latent dimension of 32) results in terms of ROC-AUC scores. The tables show different results based on combinations of batch-size, noise power and dropout probability in each model. Left: Results for discrimination based on the same filters used in training. Right: Results for discrimination using only *Error* and *Inactive* statuses.

In the tables below, all model results are shown at varying batch size, dropout for models with latent space at 32 for FNN (Table 4.11), LSTM (Table 4.12) and GRU (Table 4.13).

		Dropout Rate		
		0%	**10%**	**20%**
Batch-6				
Noise	**NO**	**0.922**	0.911	0.9071
	5%	0.91	0.91	0.9159
	10%	0.9105	0.9138	0.9111
	20%	0.9135	0.915	0.9002
Batch-12				
Noise	**NO**	**0.9197**	0.9121	0.906
	5%	0.9184	0.9119	0.9084
	10%	0.9105	0.9041	0.9029
	20%	0.9087	0.9054	0.9026

		Dropout Rate		
		0%	**10%**	**20%**
Batch-6				
Noise	**NO**	**0.9457**	0.9354	0.9431
	5%	0.937	0.9323	0.9404
	10%	0.948	0.9447	0.942
	20%	0.9431	0.9383	0.9369
Batch-12				
Noise	**NO**	**0.9512**	0.9439	0.936
	5%	0.9484	0.9373	0.9293
	10%	0.9413	0.9429	0.9333
	20%	0.9365	0.9479	0.9359

Table 4.11: FNN AE results in terms of ROC-AUC scores. Left: Results for discrimination based on the same filters used intraining. Right: Results for discrimination using only *Error* and *Inactive* statuses.

		Dropout Rate		
		0%	**10%**	**20%**
Batch-6				
Noise	**NO**	0.9138	0.9034	0.9128
	5%	0.913	0.9192	**0.9308**
	10%	0.9054	0.9294	0.9253
	20%	0.9182	0.9146	0.9239
Batch-12				
Noise	**NO**	0.9112	0.9052	0.9105
	5%	0.9164	0.9192	**0.9298**
	10%	0.9058	0.9253	0.9232
	20%	0.9184	0.9130	0.9210

		Dropout Rate		
		0%	**10%**	**20%**
Batch-6				
Noise	**NO**	0.9553	0.9478	0.9464
	5%	0.947	0.9546	**0.9637**
	10%	0.9556	0.9548	0.9600
	20%	0.9546	0.9548	0.9549
Batch-12				
Noise	**NO**	0.9518	0.9512	0.9557
	5%	0.9552	0.9545	0.9593
	10%	0.9518	0.9511	0.9504
	20%	**0.9613**	0.9557	0.9545

Table 4.12: LSTM-AE results in terms of ROC-AUC scores. Left: Results for detection based on the same filters used in training. Right: Results for detection using only *Error* and *Inactive* statuses.

		Dropout Rate		
		0%	**10%**	**20%**
Batch-6				
Noise	**NO**	0.9116	0.9033	**0.929**
	5%	0.9121	0.9166	0.9109
	10%	0.8948	0.9109	0.9157
	20%	0.9183	0.9225	0.9126
Batch-12				
Noise	**NO**	0.9106	0.8891	**0.9273**
	5%	0.9091	0.9134	0.9102
	10%	0.906	0.9183	0.9100
	20%	0.9062	0.9124	0.9045

		Dropout Rate		
		0%	**10%**	**20%**
Batch-6				
Noise	**NO**	0.9542	0.9486	**0.9559**
	5%	0.9528	0.9501	0.9508
	10%	0.952	0.9495	0.9539
	20%	0.9523	0.9505	0.9414
Batch-12				
Noise	**NO**	0.9512	0.9505	0.9524
	5%	0.9505	**0.9525**	0.9438
	10%	0.9509	0.9478	0.9493
	20%	0.9514	0.9485	0.9469

Table 4.13: GRU-AE results in terms of ROC-AUC scores. Left: Results for detection based on the same filters used in training. Right: Results for detection using only *Error* and *Inactive* statuses.

Instead, Table (4.14) show the various results of the models with VAE-type regularization.

	VAE Model		
	FNN	**LSTM**	**GRU**
Batch-6	0.8821	0.9193	0.9003
Batch-12	0.8832	**0.9196**	0.9008

	VAE Model		
	FNN	**LSTM**	**GRU**
Batch-6	0.9386	**0.9587**	0.9528
Batch-12	0.9397	0.9496	0.9514

Table 4.14: VAEs results in terms of ROC-AUC scores. The tables show different results based on combinations of batch size and network. Left: Results for discrimination based on the same filters used in training. Right: Results for detection using only *Error* and *Inactive* statuses.

	Variance		
	0.8	**0.9**	**0.95**
PCA	.8112	0.8234	**0.8423**

	Variance		
	0.8	**0.9**	**0.95**
PCA	.8576	0.8533	**0.8831**

Table 4.15: PCA results in terms of ROC-AUC scores based on principal component variance. Left: Results for discrimination based on the same filters used in training. Right: Results for detection using only *Error* and *Inactive* statuses.

Table 4.15 show the various results for the PCA.

At the end of the experiments, all models obtain rather similar results at this stage. However, the models with higher latent space obtain higher overall. In addition, RNN-type models benefit more from regularization as models that are more prone to overfitting because they contain more parameters. The batch size, does not particularly alter the results, so it can be set at the value of 12 for later experiments. By increasing the batch value, the results tend to become worse and produce mixed results.

The best model tested at this stage results in both experiments LSTM-DAE with batch size at 6 noise variance at 5% and dropout at 20%.

4.4.2 Benchmarks Validation

In the evaluation that will be done using benchmarks, the various models will be put to use in the process of detecting particular anomalies that have been artificially produced by the insertion of noise or the alteration of certain measurements that are linked with specific subsystems.

The accomplishment of these benchmarks will begin with the production of a modified signal, which will be accomplished by starting with the initial signal and then introducing modifications. Timestamps will be replaced in the original time sequences by the use of a binary mask, which will be computed through the application of probabilistic methods. But, because of the chaotic nature of the WTG systems, it will first be necessary to guarantee that the original signal is free of any earlier anomalies. This will be necessary to do because of the nature of the WTG systems. In the event that it is not, there is the potential for the introduction of anomalies into a time series that already contains further anomalies that have not been reported.

During this phase, the data from the four validation turbines are initially filtered to produce time sequences that have the highest possible level of performance. Table 4.16 outlines the many filters that were applied. After the filtering the top performance sequences are extracted, the scarcity of data with such characteristics can be seen in Table 4.17. Unlike previous experiments, the benchmarks will be evaluated on single timesteps, due to the short-lived nature of the anomaly phenomena reproduced.

Measure	**Lower Bound**	**Upper Bound**	**Other Conditions**
Active Power	-50kW	2300kW	
Wind Speed	3.5m/s	25m/s	
Performance (G90)	90%	110%	
ABS Temp. Difference		8C°	
Status			Not error status
Perf. (G90) Cent. MA 12	95%	105%	
Limitations			Not a limitation

Table 4.16: Types of filters used in the benchmark preprocessing phase. The table shows the list of quantities used and the appropriate filter conditions. All data less than the lower bound or greater than the upper bound were removed. Since *Status* is a symbolic data type the *Error Status* is removed. It is important to note that the performance settings that are employed at this step of model training are more restrictive than those that were used in the filter phase in order to generate a time series that is of the highest possible quality.

WTG	**N° Bench. Seqs.**	**% Bench. Seqs.**	**% Valid T.**
WTG-1	99/13212	0.74%	5.54%
WTG-2	1396/13212	10.5%	15.02%
WTG-3	155/13212	1.17%	8.47%
WTG-4	219/13212	1.66%	11.03%

Table 4.17: Data samples (in sequence format) after filtering and preprocessing for the benchmark testing. The fact that the number of valid sequences does not equal the number of valid timestamps resides in the fact that only sequences solely consisting of valid data are usable in the benchmark testing phase.

One way to insert anomalies is to use a masking technique where a random subset of the time series is selected, and the values in that subset are replaced with anomalous values. In this scenario, the mask is formed by employing a random chance of 1/12. This suggests that one of every twelve timesteps will be chosen to be replaced with an anomalous value in the course of the process. A value of one in the mask indicates that the corresponding timesteps in the time series should be replaced with an anomalous value, while a value of zero indicates that the timesteps should be left unmodified. The mask is a binary vector that is the same length as the time series.

In the next experiments only few measures are selected for the benchmark evalutation, with particular values of anomaly injected:

- **Active Power**: The anomaly correspond to a scaling by 0.90 of the observed *Active Power* measure. Benchmark inspired by literature on WTG registered anomalies [60].
- **Rotor RPM**: The anomaly correspond to a scaling by 1.10 of the observed *Rotor RPM* measure. Benchmark inspired by literature on WTG registered anomalies [60].
- **Gearbox Oil Temperature**: The anomaly correspond to an addictive noise Gaussian noise with zero-means and unitary variance. The noise is added to the scaled measure so the variance can be considered equal to to the one of the observed *Gearbox Oil Temperature* measure.

Using a masking technique in which a random portion of the time series is chosen, and then having the values in that portion of the time series replaced with anomalous values, is one method for introducing anomalies into the data. After the mask has been created, it is put to use to introduce anomalies into the time series. This is done by selecting the timesteps in the mask that correlate to the 1s and then changing the values of those timesteps with values that are anomalous. The random noise, spikes, trends, or any other patterns that can be important to the application are some of the types of anomalies that can be detected using the anomalous values, which can be selected according on the sort of anomaly that one wishes to identify.

Anomalies can be introduced into a time series in different ways depending on the type of anomaly being simulated. For instance, to simulate a scaling anomaly, the value of the i-th timestamp can be substituted using the mask, which is generated as a binary vector of length N, where N is the number of timestamps in the time series. The mask contains ones at random positions with probability p, where p is the proportion of anomalous timestamps. Thus, the mask M can be defined as follows:

$$M_i = \begin{cases} 1, & \text{with probability } p \\ 0, & \text{with probability } 1-p \end{cases}$$

To introduce the anomaly, the original value x_i at the i-th timestamp is multiplied by a scaling factor s, which can be randomly generated or fixed, and then replace it with the new value x'_i, as follows:

$$x'_i = \begin{cases} s \times x_i, & \text{if } M_i = 1 \\ x_i, & \text{otherwise} \end{cases}$$

On the other hand, to simulate an additive anomaly, such as spikes or noise, the pattern can be added directly to the original time series using the mask. For example, a spike anomaly can be generated by adding a random value a to the i-th timestamp, which is selected by the mask. Thus, the mask A can be defined as follows:

$$A_i = \begin{cases} a, & \text{with probability } p \\ 0, & \text{with probability } 1-p \end{cases}$$

To introduce the anomaly, the anomaly mask A is added to the original time series x, as follows:

$$x'_i = x_i + A_i$$

By using these techniques, different types of anomalies can be artificially introduced into a time series and use them to evaluate the performance of anomaly detection algorithms.

Measured Active Power Benchmark

Below are the results for the anomaly detection phase on the *Active Power* benchmark. It can be observed how the recurrent models are able to produce excellent results. The feed-forward type models, while achieving good results, have significantly lower scores.

Table 4.18, 4.19, 4.20 and 4.21 show in that order, the results of FFN, LSTM, GRU and the different VAEs. The tables show the ROC-AUC score values at varying hyperparameters of noise variance (in the AE denoise problem) and dropout probability. Since results are almost identical, table reports only batch sizes of 12.

		Dropout Rate	
		0%	**10%**
Noise	**NO**	**0.8478**	0.8157
	5%	0.8157	0.783
	10%	0.8294	0.8163
	20%	0.8352	0.7744

		Dropout Rate		
		0%	**10%**	**20%**
Noise	**NO**	**0.8514**	0.7528	0.8029
	5%	0.8267	0.835	0.8667
	10%	0.8363	0.8277	0.8277
	20%	0.8412	0.8404	0.7914

Table 4.18: FNN AE results in terms of ROC-AUC scores for the *Active Power* benchmark. Left: latent dimension 32. Right: latent dimension 48.

		Dropout Rate	
		0%	**10%**
Noise	**NO**	0.8932	0.8932
	5%	**0.9071**	0.8915
	10%	0.8921	0.8963
	20%	0.9058	0.8966

		Dropout Rate		
		0%	**10%**	**20%**
Noise	**NO**	0.9286	0.9362	0.9329
	5%	0.9284	0.9305	0.9178
	10%	0.9377	**0.9391**	0.9218
	20%	0.9367	0.9339	0.9334

Table 4.19: LSTM-AE results in terms of ROC-AUC scores for the *Active Power* benchmark. Left: latent dimension 32. Right: latent dimension 48.

		Dropout Rate	
		0%	**10%**
Noise	**NO**	0.9024	0.8914
	5%	0.9042	0.8905
	10%	0.8983	0.8549
	20%	**0.9034**	0.8549

		Dropout Rate		
		0%	**10%**	**20%**
Noise	**NO**	0.9255	0.9183	0.9099
	5%	0.9262	0.931	0.9262
	10%	0.9365	0.9358	0.9293
	20%	0.9281	**0.9394**	0.9311

Table 4.20: GRU-AE results in terms of ROC-AUC scores for the *Active Power* benchmark. Left: latent dimension 32. Right: latent dimension 48.

	VAE Model		
	FNN	**LSTM**	**GRU**
Batch-12	0.8634	0.8832	**0.8848**

	VAE Model		
	FNN	**LSTM**	**GRU**
Batch-12	0.8723	**0.9029**	0.8937

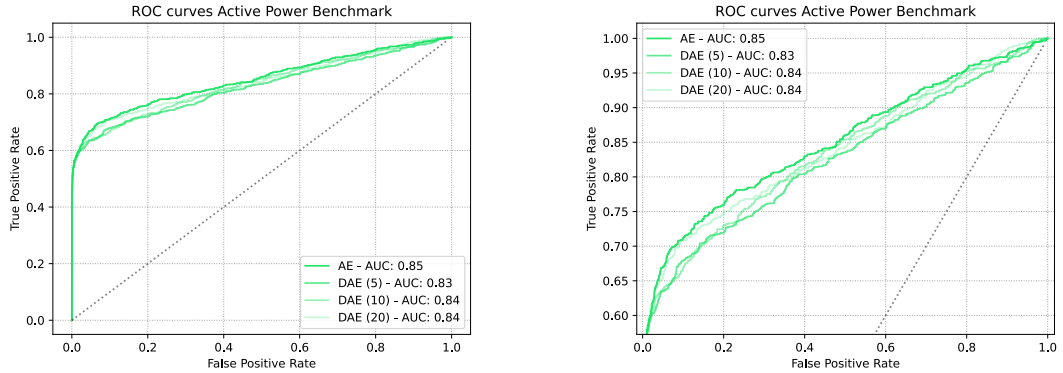
Table 4.21: VAEs results in terms of ROC-AUC scores for the *Active Power* benchmark. Left: latent dimension 32. Right: latent dimension 48.

	Variance		
	0.8	**0.9**	**0.95**
PCA	.7932	0.8056	**0.8239**

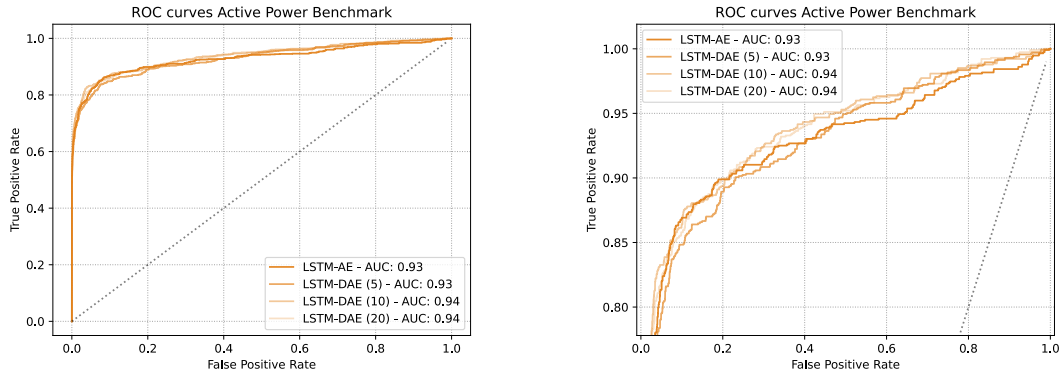
Table 4.22: PCA results in terms of ROC-AUC scores for the *Active Power* benchmark.

Table 4.22 show the various results for the PCA.

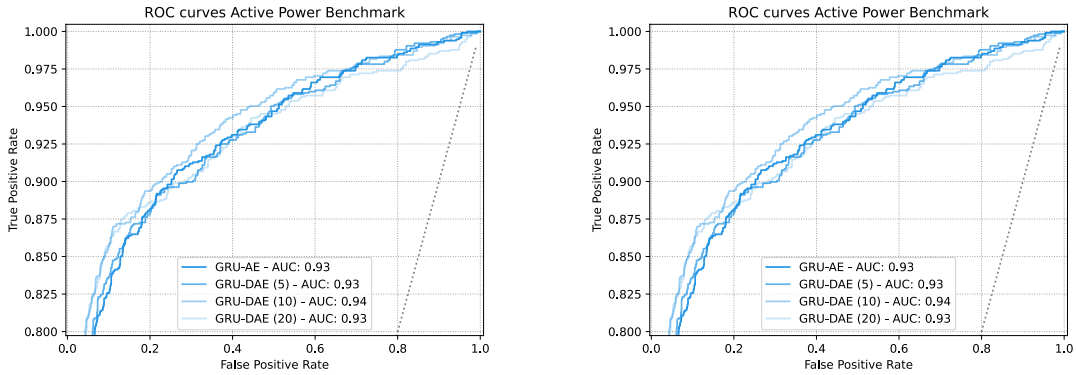
Furthermore, some ROC curves for the most important models that were employed are depicted in Figure 4.22 that may be found below.



((a)) ROC curves for AE.



((b)) ROC curves for LSTM.



((c)) ROC curves for GRU.

Figure 4.22: ROC curves for the *Rotor RPM* benchmark for AE and DAE models. Left: whole curve. Right: curve zoomed on high TPR. Only for latent dimension 48.

Gearbox Oil Temperature Benchmark

The outcomes of the anomaly detection phase on the *Gearbox Oil Temperature* benchmark are included in the following. Once again, recurrent patterns produce superior results on this detection stage. This time, however, the results are slightly worse than the proceeding ones; the case becomes much worse when feed-forward models are used. The main reason for that may be:

- **Dependence between variables**: In a multivariate time series, the variables are often interdependent, and anomalies in one variable can affect the behavior of other variables. This dependence can make it harder to detect anomalies because the anomaly signal may be spread across multiple variables and may not be as pronounced as in a univariate time series.
- **Noise in the normal measure**: Even if there is more noise in the normal measure of a multivariate time series, this may not necessarily make it easier to detect anomalies. In fact, if the noise is already high, the anomaly signal may be masked by the noise, making it harder to detect. Moreover, the noise may also have a similar statistical distribution as the anomalies, making it harder to distinguish between the two.

Table 4.23, 4.24, 4.25 and 4.26 show in that order, the results of FFN, LSTM, GRU and VAEs. The tables show the ROC-AUC score values at varying hyperparameters of noise variance (in the autoencoder denoise problrm) and dropout probability. Since results are almost identical, table reports only batch sizes of 12.

		Dropout Rate	
		0%	**10%**
Noise	**NO**	0.7487	0.6392
	5%	0.6392	0.7413
	10%	0.7273	**0.7192**
	20%	0.7101	0.7132

		Dropout Rate		
		0%	**10%**	**20%**
Noise	**NO**	0.6732	0.7201	0.7813
	5%	0.6971	0.7825	**0.8023**
	10%	0.7478	0.7466	0.7466
	20%	0.6623	0.7282	0.7984

Table 4.23: FNN AE results in terms of ROC-AUC scores for the *Gearbox Oil Temperature* benchmark. Left: latent dimension 32. Right: latent dimension 48.

		Dropout Rate	
		0%	**10%**
Noise	**NO**	0.8132	0.8132
	5%	**0.8349**	0.804
	10%	0.7919	0.8165
	20%	0.8267	0.8077

		Dropout Rate		
		0%	**10%**	**20%**
Noise	**NO**	0.868	0.8711	0.8493
	5%	0.8741	0.867	0.8485
	10%	0.8736	**0.874**	0.8599
	20%	0.8693	0.8534	0.8527

Table 4.24: LSTM-AE results in terms of ROC-AUC scores for the *Gearbox Oil Temperature* benchmark. Left: latent dimension 32. Right: latent dimension 48.

		Dropout Rate	
		0%	**10%**
Noise	**NO**	0.8367	0.8186
	5%	**0.8375**	0.8186
	10%	0.8231	0.7841
	20%	0.8215	0.7841

		Dropout Rate		
		0%	**10%**	**20%**
Noise	**NO**	0.8501	0.8601	0.8364
	5%	0.8549	0.8682	0.8584
	10%	**0.8801**	0.8624	0.8544
	20%	0.8628	0.8575	0.8539

Table 4.25: GRU-AE results in terms of ROC-AUC scores for the *Gearbox Oil Temperature* benchmark. Left: latent dimension 32. Right: latent dimension 48.

Table 4.27 show the various results for the PCA.

ROC curves for the most important models that were employed are depicted in Figure 4.23 that may be found below.

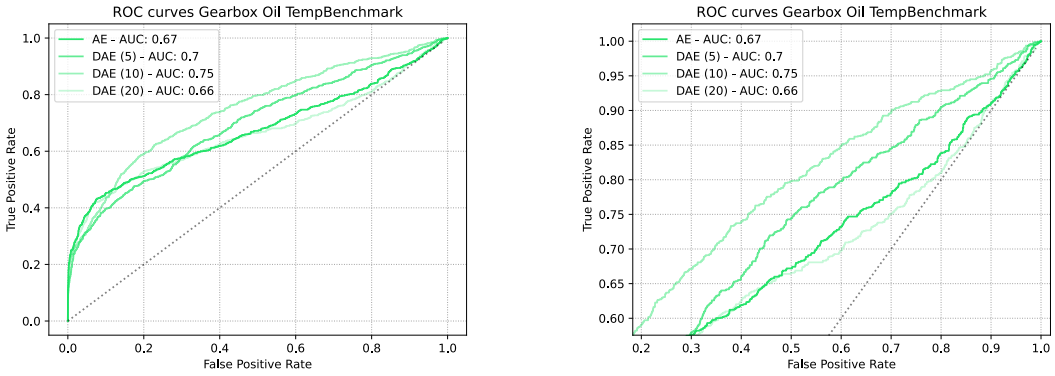
	VAE Model		
	FNN	**LSTM**	**GRU**
Batch-12	0.7323	**0.7456**	0.7453

	VAE Model		
	FNN	**LSTM**	**GRU**
Batch-12	0.7566	**0.7717**	0.748

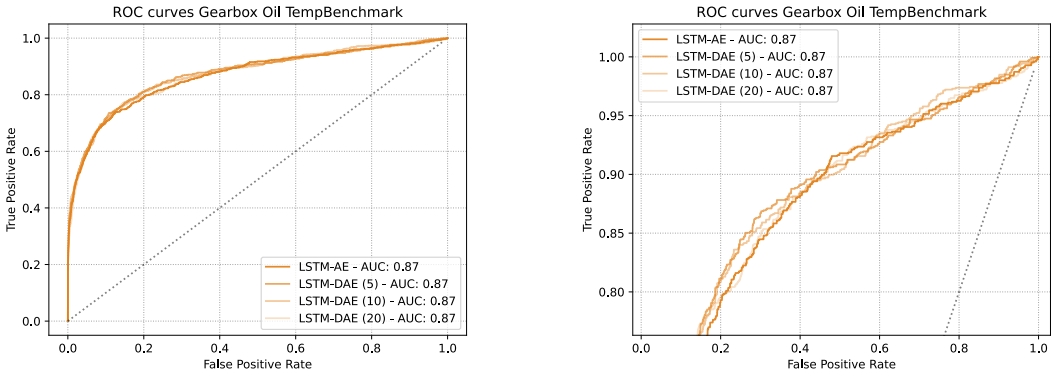
Table 4.26: VAEs results in terms of ROC-AUC scores for the *Gearbox Oil Temperature* benchmark. Left: latent dimension 32. Right: latent dimension 48.

	Variance		
	0.8	**0.9**	**0.95**
PCA	.7123	0.7345	x**0.7717**

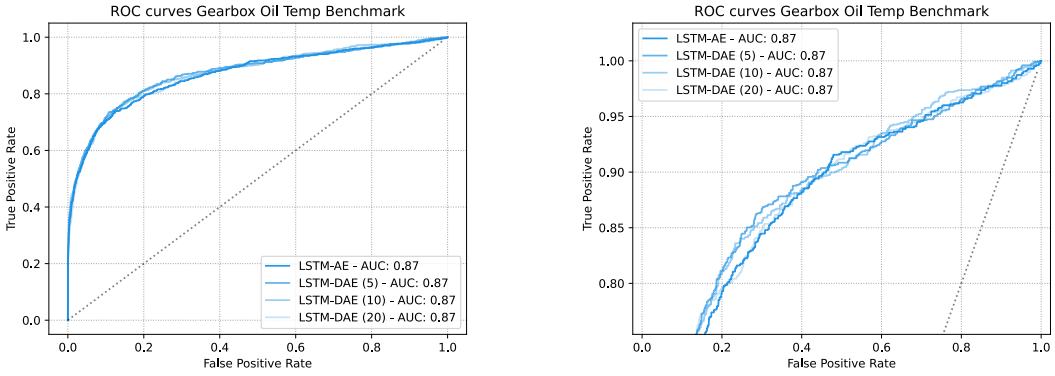
Table 4.27: PCA results in terms of ROC-AUC scores for the *Gearbox Oil Temperature* benchmark.



((a)) ROC curves for AE.



((b)) ROC curves for LSTM.



((c)) ROC curves for GRU.

Figure 4.23: ROC curves for the *Active Power* benchmark for AE and DAE models. Left: whole curve. Right: curve zoomed on high TPR. Only for latent dimension 48.

Rotor RPM Benchmark

Below are the results for the anomaly detection phase on the *Rotor RPM* benchmark. It is clear that recurrent models are capable of generating high-quality output, as can be shown above. Just like the previous case, even though they produce satisfactory outcomes, feed-forward models have much lower score averages.

Table 4.28, 4.29 and 4.30 and 4.31 show in that order, the results of FFN, LSTM, GRU and VAE. The tables show the ROC-AUC score values at varying hyperparameters of noise variance (in the AE denoise problem) and dropout probability. Only batch 12 results are shown.

		Dropout Rate	
		0%	**10%**
Noise	**NO**	0.8314	0.8177
	5%	0.8177	0.8209
	10%	0.8376	**0.8375**
	20%	0.8361	0.8302

		Dropout Rate		
		0%	**10%**	**20%**
Noise	**NO**	0.8165	0.8973	0.9265
	5%	0.8052	0.9107	**0.9403**
	10%	0.8131	0.9046	0.9046
	20%	0.8047	0.9286	0.9056

Table 4.28: FNN AE results in terms of ROC-AUC scores for the *Rotor RPM* benchmark. Left: latent dimension 32. Right: latent dimension 48.

		Dropout Rate	
		0%	**10%**
Noise	**NO**	0.9295	0.9182
	5%	**0.9353**	0.9234
	10%	0.9308	0.9297
	20%	0.9269	0.9238

		Dropout Rate		
		0%	**10%**	**20%**
Noise	**NO**	0.9698	0.9748	0.9756
	5%	0.973	0.9716	0.9597
	10%	0.969	0.9745	**0.9759**
	20%	0.9756	0.9686	0.9718

Table 4.29: LSTM-AE results in terms of ROC-AUC scores for the *Rotor RPM* benchmark. Left: latent dimension 32. Right: latent dimension 48.

		Dropout Rate	
		0%	**10%**
Noise	**NO**	0.9379	0.926
	5%	0.9398	0.931
	10%	0.9348	0.8909
	20%	**0.9417**	0.8909

		Dropout Rate		
		0%	**10%**	**20%**
Noise	**NO**	0.9697	0.9638	0.9723
	5%	0.9727	0.9754	0.9624
	10%	0.9626	**0.9759**	0.9634
	20%	0.9731	0.9607	0.9681

Table 4.30: GRU-AE results in terms of ROC-AUC scores for the *Rotor RPM* benchmark. Left: latent dimension 32. Right: latent dimension 48.

	VAE Model		
	FNN	**LSTM**	**GRU**
Batch-12	0.8634	0.8832	**0.8848**

	VAE Model		
	FNN	**LSTM**	**GRU**
Batch-12	0.8723	**0.9029**	0.8937

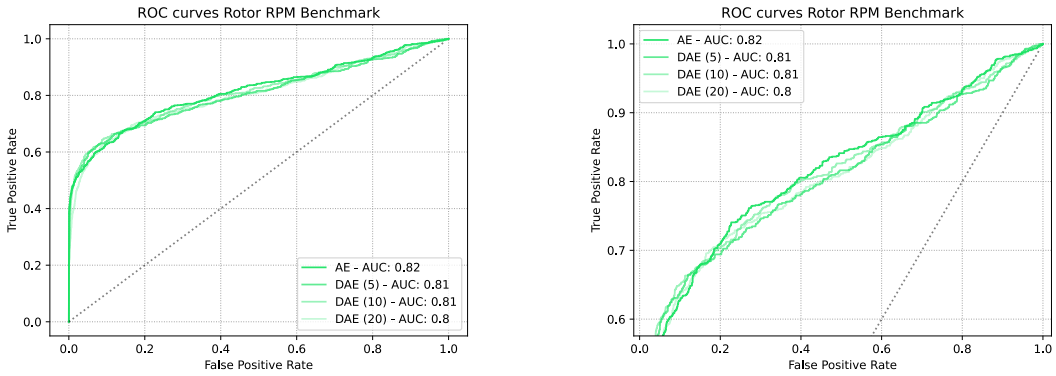
Table 4.31: VAEs results in terms of ROC-AUC scores for the *Rotor RPM* benchmark. Left: latent dimension 32. Right: latent dimension 48.

	Variance		
	0.8	**0.9**	**0.95**
PCA	.7653	0.7867	**0.7967**

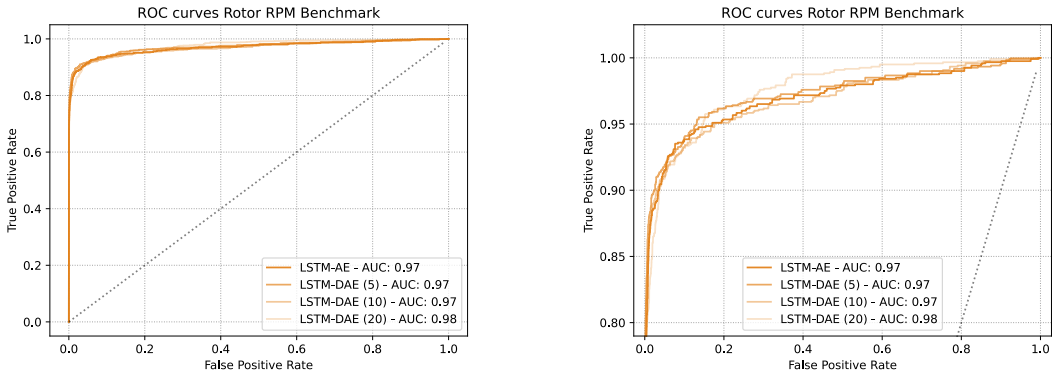
Table 4.32: PCA results in terms of ROC-AUC scores for the *Rotor RPM* benchmark.

Table 4.32 show the various results for the PCA.

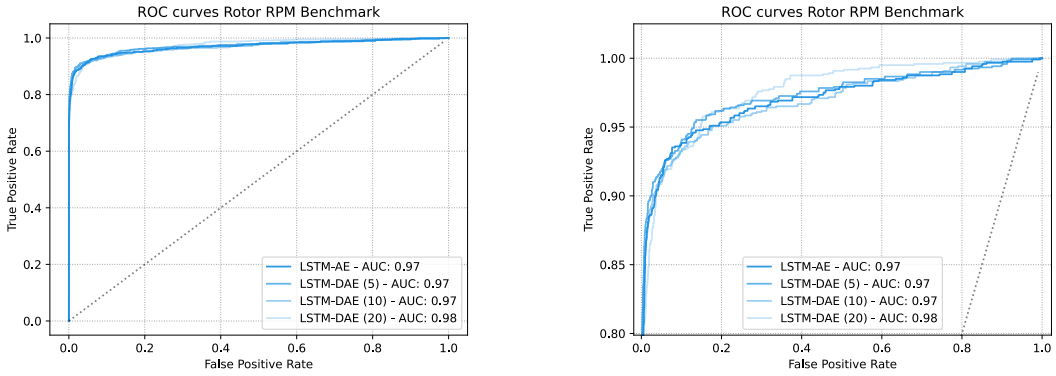
Again, some ROC curves for the most important models that were employed are depicted in Figure 4.24 that may be found below.



((a)) ROC curves for AE.



((b)) ROC curves for LSTM.



((c)) ROC curves for GRU.

Figure 4.24: ROC curves for the *Active Power* benchmark for AE and DAE models. Left: whole curve. Right: curve zoomed on high TPR. Only for latent dimension 48.

4.4.3 Downtimes Detection Validation

In this specific instance, there is only one downtime associated with one of the turbines that can be researched within the validation set.

- **WTG-3**:
  - **2021/08/31**: *Generator bearing high temperature* from 02:15:28 to 02:33:07.
  - **2021/08/31**: *Generator bearing high temperature* from 16:47:42 to 17:00:53.

WTG-3 Generator Bearing High Temperature

The Figure 4.25 displays the error in the reconstruction of the most important parameters during this downtime; from the image, it is clear that a number of measurements are contributing to a significant deviation. Nevertheless, the measure that has the most obvious impact in the downtime is is still the *Generator Bearing High Temperature*.

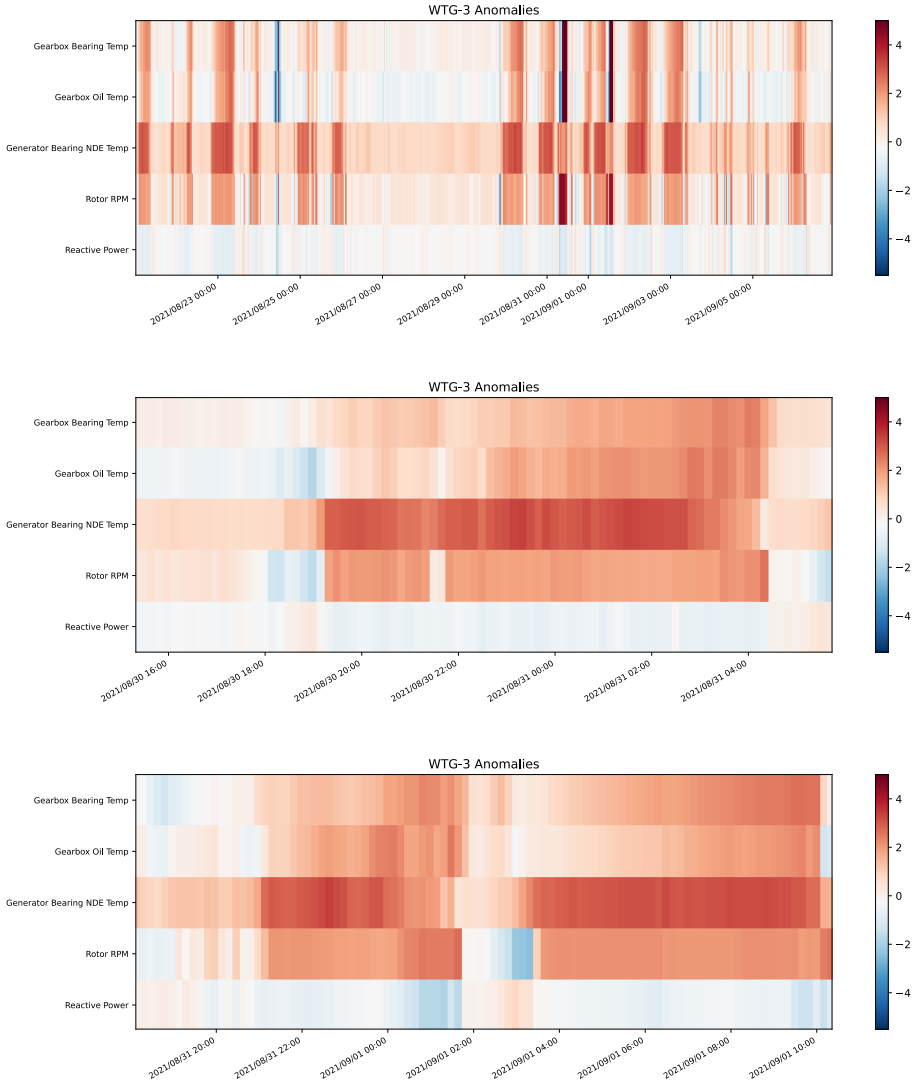


Figure 4.25: Heatmap of the RE for the most significant measures of the *Generator Bearing High Temperature* downtime.

The correlation between the anomalies found can be explained by the fact that through the generator shaft, the rotational energy that is generated by the rotor is transferred to the generator.

The generator bearing is connected to the rest of the system through the generator shaft. The rotational speed of the rotor is typically increased by a gearbox that is attached to the generator shaft. This ensures that the rotational speed of the rotor is compatible with the requirements of the generator. In the event that the bearing fails, it is possible that the rotor would become frozen and stop rotating, which will lead to expensive downtime and repairs.

Anomalies on the generator bearing can be propagated through the system in several ways. For example, if the bearing becomes worn or damaged, it may produce increased levels of vibration that can be transmitted through the generator shaft and into the gearbox and other components in the drivetrain. These vibrations can cause additional wear and tear on these components and increase the risk of failure.

In the Tables 4.33, 4.34, 4.35 the results for the different models in the determination of error statuses and stopping moments are shown. In the next tables batch with size 12 and latent space of size 48 models results are shown.

		Dropout Rate		
		0%	**10%**	**20%**
Noise	**NO**	**0.9234**	0.9122	0.9095
	5%	0.9112	0.9047	0.9049
	10%	0.9101	0.9100	0.9022
	20%	0.911	0.9002	0.9066

Table 4.33: FNN AE results in terms of ROC-AUC scores for the 10 days surrounding the *Generator Bearing High Temperature* downtime.

		Dropout Rate		
		0%	**10%**	**20%**
Noise	**NO**	0.9355	0.9214	0.9253
	5%	0.9125	0.9206	0.9221
	10%	0.9359	0.9489	0.9487
	20%	0.9408	**0.9502**	0.9243

Table 4.34: LSTM-AE results in terms of ROC-AUC scores for the 10 days surrounding the *Generator Bearing High Temperature*.

		Dropout Rate		
		0%	**10%**	**20%**
Noise	**NO**	0.9334	0.9235	0.9145
	5%	0.9234	0.9213	0.9255
	10%	0.9453	0.9519	0.9481
	20%	0.9458	**0.9522**	0.9321

Table 4.35: GRU-AE results in terms of ROC-AUC scores for the 10 days surrounding the *Generator Bearing High Temperature* downtime.

4.4.4 Event Discrimination Testing

A Status discrimination is performed as an initial test to the new set of turbines.

At this point in the process, testing data has been labeled using the same labels that were used to discriminate data for training dataset. As the previous experiment, RE is aggregated in the sequence with an MSE (between all measures and timesteps inside the sequence). The labels referring to an anomalous sequence are obtained by aggregating the single timesteps labels using a logical-OR between all the single timesteps labels, plus a performance average between all the timestep of a sequence.

Also this time a partial method is used, in which only particular subsets of labels are to be evaluated. The requirement here is that the subsets in question contain partitions with higher reconstruction error, or that they only use partitions with both positive and negative labels. In this partition test, only few filter conditions are considered.

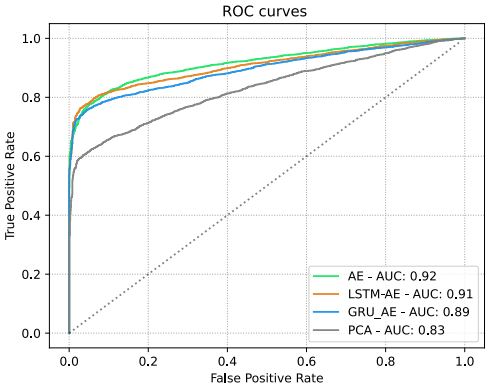


Figure 4.26: ROC curves classifications results with the main models. The same filters used in training are used for labelling the data.

In Figure 4.26, it is clear that every model has the potential to readily deliver excellent outcomes in the classification phase by making an effort to positively label the data in accordance with the splits that were utilized in the filters during the training phase.

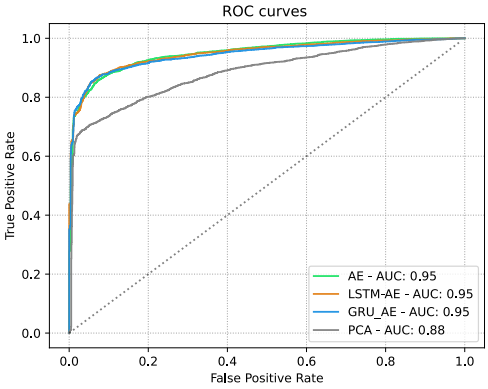
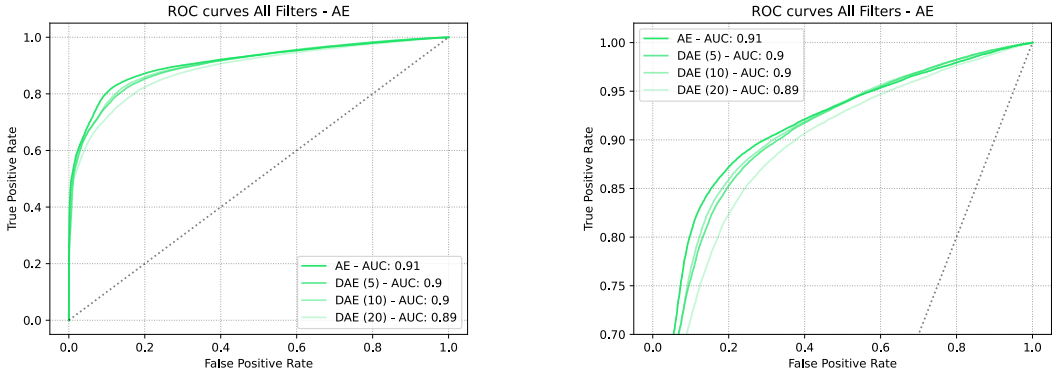
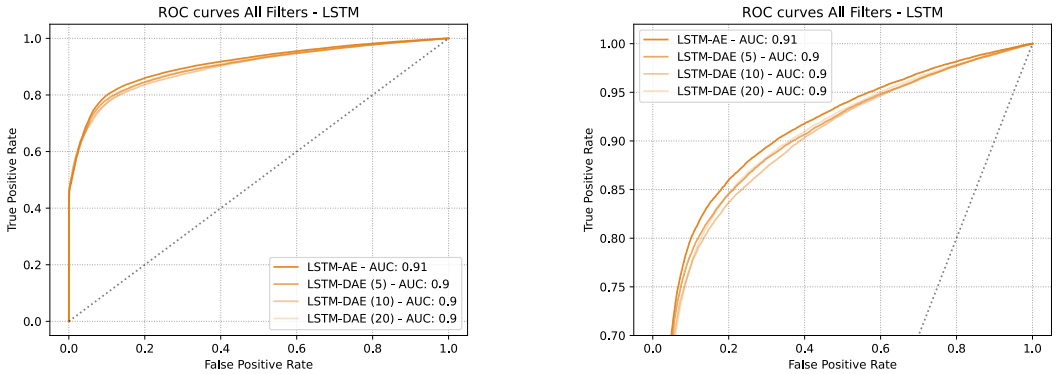


Figure 4.27: ROC curves classifications results with the main models. The same filters used in training are used for labelling the data.

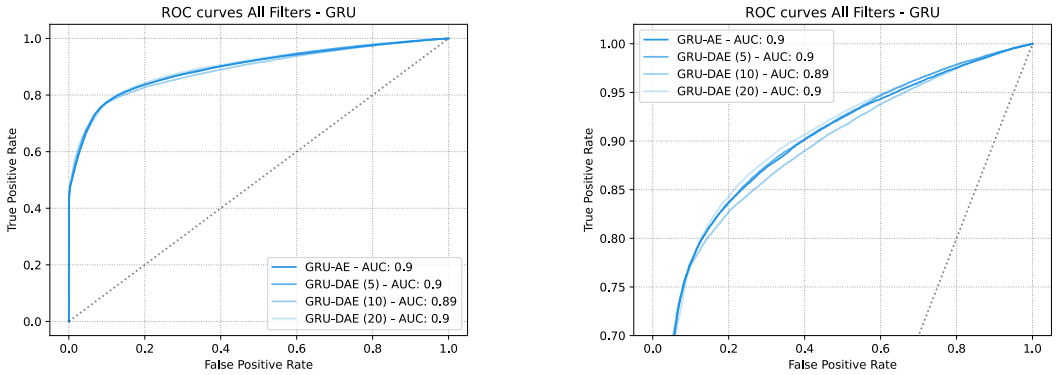
In Figure 4.27, on the other hand, it is possible to visualize the same patterns in the classification stage using the turbine *Status* in the observed timestamp as the discriminating label.



((a)) ROC curves for AE.



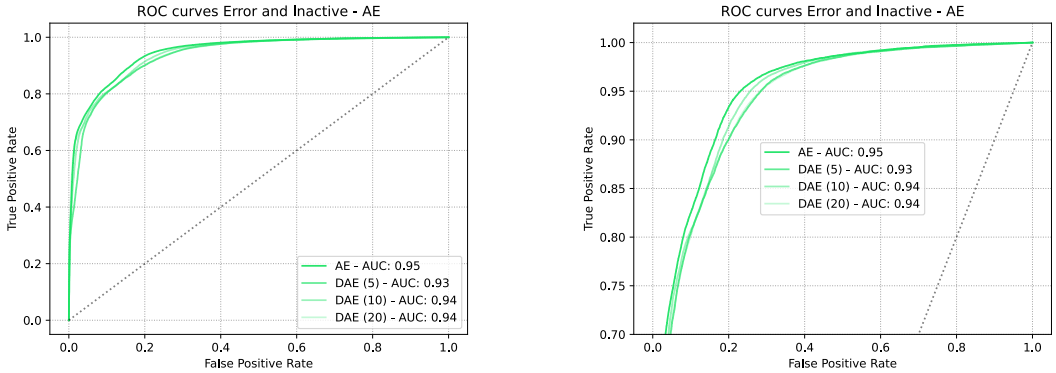
((b)) ROC curves for LSTM.



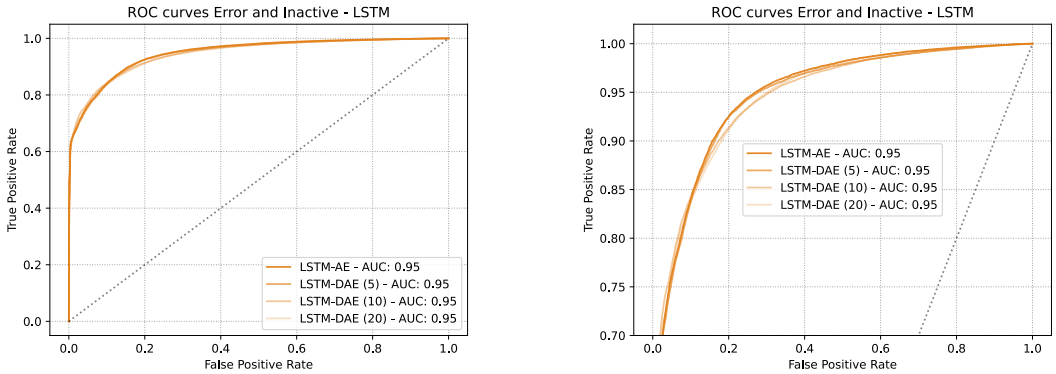
((c)) ROC curves for GRU.

Figure 4.28: ROC curves for all filters involve in training for AE and DAE models. Left: whole curve. Right: curve zoomed on high TPR.

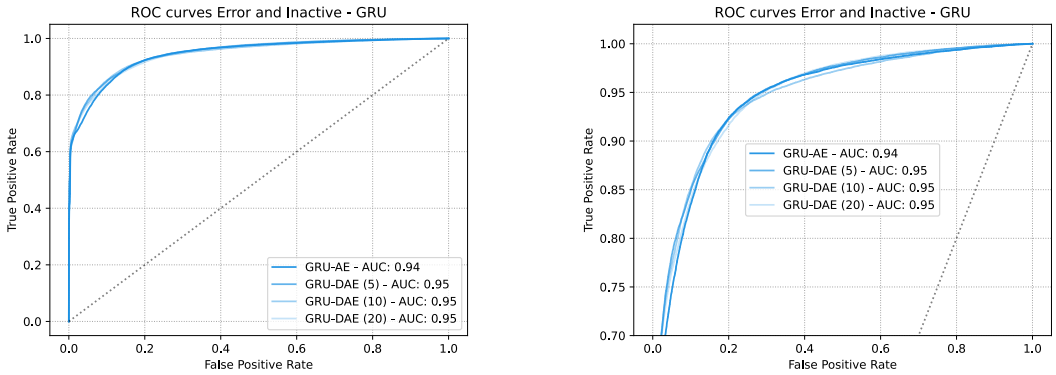
In Figure 4.28, it can be observed that, it is possible for any DAE model, whether it be a FNN or an RNN, to acquire very high TPR values despite having very low FPR values.



((a)) ROC curves for AE.



((b)) ROC curves for LSTM.



((c)) ROC curves for GRU.

Figure 4.29: ROC curves for all filters involve in training for AE and DAE models. Left: whole curve. Right: curve zoomed on high TPR.

From Figure 4.29 it can be observed, all DAE models, both FNN and RNN, are able to achieve very high TPR values with very low FPR values.

In general, the same outcomes are observed in these tests as in the validation dataset testing phase. The results reaffirm that RNN models are superior, despite the fact that the scores differ by just a minor amount. Regularization benefits RNN models, however FNN models perform better when this is decreased. On general, the scores are lower, but the difference is negligible given that these data originate from turbines that have not yet been observed.

In following tables (4.36, 4.37 and 4.38) results are shown for models with latent space dimension at 32.

		Dropout Rate	
		0%	**10%**
Batch-12			
Noise	**NO**	0.8928	**0.8980**
	5%	0.8918	0.8818
	10%	0.8937	0.8877
	20%	0.8910	0.8819

		Dropout Rate	
		0%	**10%**
Batch-12			
Noise	**NO**	**0.9469**	0.9432
	5%	0.9387	0.9304
	10%	0.9441	0.9394
	20%	0.9351	0.9317

Table 4.36: FNN AE (latent dimension of 32) results in terms of ROC-AUC scores. Left: Results for discrimination based on the same filters used in training. Right: Results for discrimination using only *Error* and *Inactive* statuses.

		Dropout Rate	
		0%	**10%**
Batch-12			
Noise	**NO**	0.8916	0.9005
	5%	0.8976	0.9046
	10%	0.8921	**0.9088**
	20%	0.8912	0.9012

		Dropout Rate	
		0%	**10%**
Batch-12			
Noise	**NO**	0.9443	0.9421
	5%	0.9447	0.948
	10%	0.942	0.**9481**
	20%	0.9432	0.9442

Table 4.37: LSTM-AE (latent dimension of 32) results in terms of ROC-AUC scores. Left: Results for discrimination based on the same filters used in training. Right: Results for discrimination using only *Error* and *Inactive* statuses.

		Dropout Rate	
		0%	**10%**
Batch-12			
Noise	**NO**	0.8875	**0.9012**
	5%	0.8944	0.8999
	10%	0.8825	0.8886
	20%	0.892	0.892

		Dropout Rate	
		0%	**10%**
Batch-12			
Noise	**NO**	0.9425	0.9421
	5%	0.9421	**0.9479**
	10%	0.9363	0.9365
	20%	0.9379	0.9379

Table 4.38: GRU-AE (latent dimension of 32) results in terms of ROC-AUC scores. Left: Results for discrimination based on the same filters used in training. Right: Results for discrimination using only *Error* and *Inactive* statuses.

In following tables (4.39, 4.40, 4.42 and 4.42) results are shown for models with latent space dimension at 48. PCA results are in Table 4.43.

		Dropout Rate		
		0%	**10%**	**20%**
Batch-12				
Noise	**NO**	**0.9103**	0.863	0.8735
	5%	0.9019	0.8799	0.886
	10%	0.9048	0.8862	0.8862
	20%	0.8883	0.8768	0.8684

		Dropout Rate		
		0%	**10%**	**20%**
Batch-12				
Noise	**NO**	**0.9478**	0.9266	0.9317
	5%	0.9346	0.9282	0.9326
	10%	0.9413	0.9356	0.9356
	20%	0.9401	0.9295	0.9285

Table 4.39: FNN AE results in terms of ROC-AUC scores. Left: Results for discrimination based on the same filters used in training. Right: Results for discrimination using only *Error* and *Inactive* statuses.

		Dropout Rate		
		0%	**10%**	**20%**
Batch-12				
Noise	**NO**	0.9091	0.9081	0.9123
	5%	0.901	0.921	0.9242
	10%	0.8975	0.9127	**0.9267**
	20%	0.9014	0.9122	0.9167

		Dropout Rate		
		0%	**10%**	**20%**
Batch-12				
Noise	**NO**	0.9486	0.9496	0.9500
	5%	0.9463	0.9479	0.9512
	10%	0.9455	0.9492	**0.9522**
	20%	0.9459	0.9434	0.9442

Table 4.40: LSTM-AE results in terms of ROC-AUC scores. Left: Results for discrimination based on the same filters used in training. Right: Results for discrimination using only *Error* and *Inactive* statuses.

		Dropout Rate		
		0%	**10%**	**20%**
Batch-12				
Noise	**NO**	0.8954	0.8944	0.8962
	5%	0.8973	0.9085	0.8923
	10%	0.8943	0.9123	0.9045
	20%	0.9043	**0.9126**	0.9042

		Dropout Rate		
		0%	**10%**	**20%**
Batch-12				
Noise	**NO**	0.9456	0.9488	0.9492
	5%	0.9443	0.9445	0.9386
	10%	0.9448	0.9502	0.9441
	20%	0.9428	**0.9505**	0.9497

Table 4.41: GRU-AE results in terms of ROC-AUC scores. Left: Results for discrimination based on the same filters used in training. Right: Results for discrimination using only *Error* and *Inactive* statuses.

	VAE Model		
	FNN	**LSTM**	**GRU**
Batch-12	0.8521	0.8642	**0.8923**

	VAE Model		
	FNN	**LSTM**	**GRU**
Batch-12	0.8942	**0.9043**	0.8921

Table 4.42: VAEs results in terms of ROC-AUC scores. Left: Results for discrimination based on the same filters used in training. Right: Results for discrimination using only *Error* and *Inactive* statuses.

	Variance		
	0.8	**0.9**	**0.95**
PCA	.8257	0.8235	**0.8342**

	Variance		
	0.8	**0.9**	**0.95**
PCA	.8563	0.8459	0.8798

Table 4.43: PCA results in terms of ROC-AUC scores based on principal component variance. Left: Results for discrimination based on the same filters used in training. Right: Results for detection using only *Error* and *Inactive* statuses.

4.4.5 Benchmarks Testing

In the benchmark testing, the various models will be utilized in the process of recognizing certain abnormalities. Those one, similarly to the evaluation ones, have been artificially manufactured by the introduction of noise or the modification of certain data associated with particular subsystems. The main difference this time is that these anomalies will not be included in test periods related only to individual months used in the validation phase. Since in the current case the test phases, take the entire year, the anomalies will be entered on a data type with a more variable seasonality.

This difficulty will be in addition to those already mentioned in the unobserved turbine testing phase previously discussed.

Measured Active Power Benchmark

Below are the results for the anomaly detection phase on the *Active Power* benchmark. Just like for the validation set of WTG, it can be seen that recurrent models are capable of producing good outcomes. While feed-forward models achieve good performance, their scores are much lower. In contrast to the validation set, some results are below the validation ones while other, the ones with high level of denoising are few points above. Those are acceptable results considering the difficulties of this dataset compared to the previous one.

Table 4.44, 4.45 and 4.46 show in that order, the results of FFN, LSTM and GRU. The tables show the ROC-AUC score values at varying hyperparameters of noise variance (in the AE denoise problem) and dropout probability

		Dropout Rate	
		0%	**10%**
Noise	**NO**	0.8324	0.7834
	5%	**0.8503**	0.7922
	10%	0.7834	0.7761
	20%	0.7521	0.7697

		Dropout Rate		
		0%	**10%**	**20%**
Noise	**NO**	**0.8475**	0.7425	0.7938
	5%	0.8249	0.8208	0.8402
	10%	0.82	0.8147	0.8147
	20%	0.8097	0.8228	0.7756

Table 4.44: FNN AE results in terms of ROC-AUC scores for the *Active Power* benchmark.

		Dropout Rate	
		0%	**10%**
Noise	**NO**	0.8982	0.8982
	5%	0.8928	0.8947
	10%	0.8904	0.9045
	20%	**0.9073**	0.8948

		Dropout Rate		
		0%	**10%**	**20%**
Noise	**NO**	**0.9436**	0.9294	0.9223
	5%	0.9324	0.9324	0.92
	10%	0.9368	0.9316	0.9283
	20%	0.9382	0.9302	0.9315

Table 4.45: LSTM-AE results in terms of ROC-AUC scores for the *Active Power* benchmark.

		Dropout Rate	
		0%	**10%**
Noise	**NO**	0.8959	**0.9042**
	5%	0.9032	0.901
	10%	0.9008	0.8773
	20%	0.9095	0.9083

		Dropout Rate		
		0%	**10%**	**20%**
Noise	**NO**	0.9276	0.9343	0.9211
	5%	0.9245	0.9391	0.9364
	10%	**0.9437**	0.9335	0.9359
	20%	0.9358	0.9405	0.9236

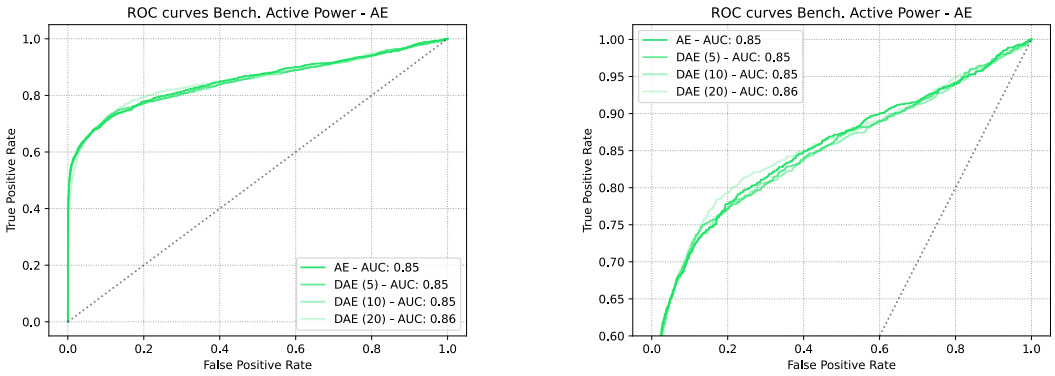
Table 4.46: GRU-AE results in terms of ROC-AUC scores for the *Active Power* benchmark.

PCA results are in Table 4.47.

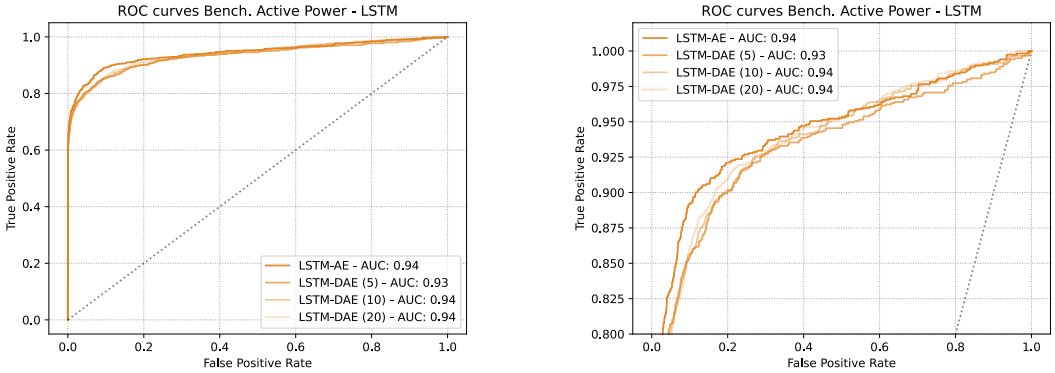
In addition, Figure 4.30 depicts ROC curves for the most significant models selected.

	Variance		
	0.8	**0.9**	**0.95**
PCA	.7743	0.7895	**0.8099**

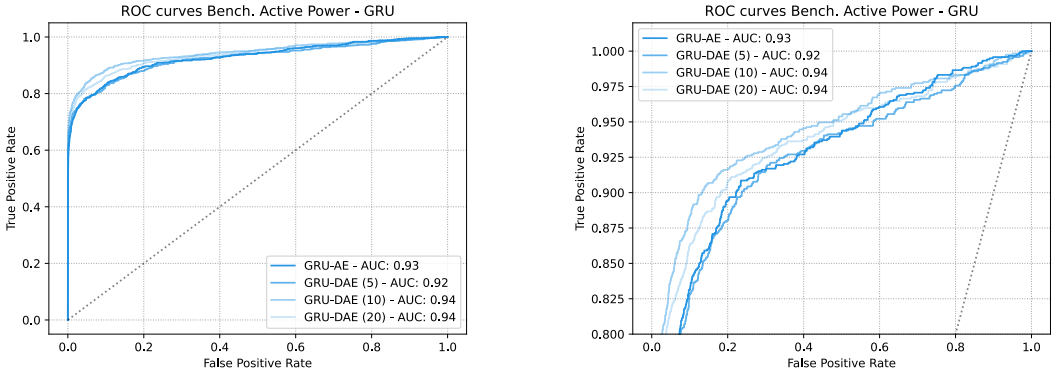
Table 4.47: PCA results in terms of ROC-AUC scores for the *Active Power* benchmark.



((a)) ROC curves for AE.



((b)) ROC curves for LSTM.



((c)) ROC curves for GRU.

Figure 4.30: ROC curves for all filters involve in training for AE and DAE models. Left: whole curve. Right: curve zoomed on high TPR.

Gearbox Oil Temperature Benchmark

The following are the outcomes of the anomaly identification phase for the *Gearbox Oil Temperature* benchmark. Again, RNN models give superior outcomes at this stage of detection. This time, however, the outcomes are marginally worse than validation phase; the situation becomes significantly better for the FNN model in this phase.

The Tables 4.48, 4.49 and 4.50 show in that order, the results of FFN, LSTM and GRU. PCA results are in Table 4.51.

		Dropout Rate	
		0%	**10%**
Noise	**NO**	0.731	0.6921
	5%	0.7155	0.713
	10%	0.6921	**0.7437**
	20%	0.6688	0.7326

		Dropout Rate		
		0%	**10%**	**20%**
Noise	**NO**	0.7953	0.7466	0.7815
	5%	0.7957	0.744	0.8137
	10%	**0.8309**	0.7584	0.7584
	20%	0.771	0.7257	0.7628

Table 4.48: FNN AE results in terms of ROC-AUC scores for the *Gearbox Oil Temperature* benchmark.

		Dropout Rate	
		0%	**10%**
Noise	**NO**	0.8193	0.8193
	5%	0.8198	0.8282
	10%	0.7968	**0.8295**
	20%	0.828	0.8263

		Dropout Rate		
		0%	**10%**	**20%**
Noise	**NO**	0.8828	0.8431	0.803
	5%	0.8719	0.8596	0.7969
	10%	0.8667	0.8414	0.8218
	20%	0.8671	0.8348	0.8367

Table 4.49: LSTM-AE results in terms of ROC-AUC scores for the *Gearbox Oil Temperature* benchmark.

		Dropout Rate	
		0%	**10%**
Noise	**NO**	**0.8295**	0.8205
	5%	0.8205	0.8285
	10%	0.8278	0.8188
	20%	0.8042	0.8226

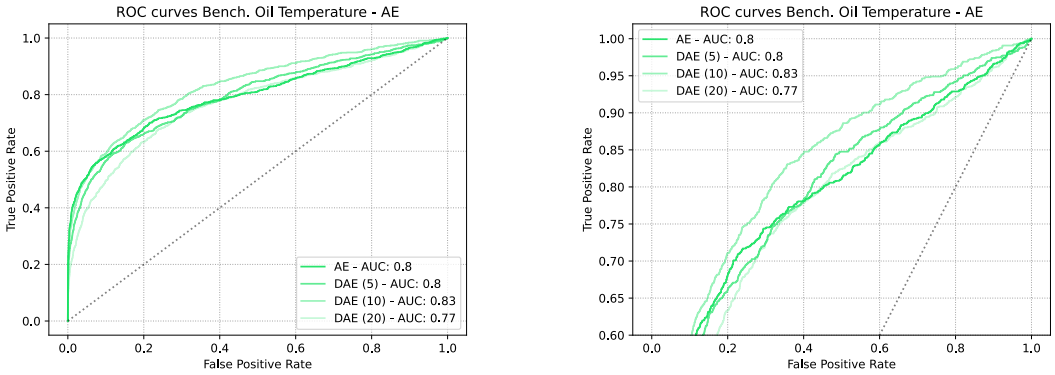
		Dropout Rate		
		0%	**10%**	**20%**
Noise	**NO**	0.8771	0.8711	0.8472
	5%	0.8774	0.8758	0.8534
	10%	**0.8777**	0.8747	0.8434
	20%	0.8729	0.8711	0.8575

Table 4.50: GRU-AE results in terms of ROC-AUC scores for the *Gearbox Oil Temperature* benchmark.

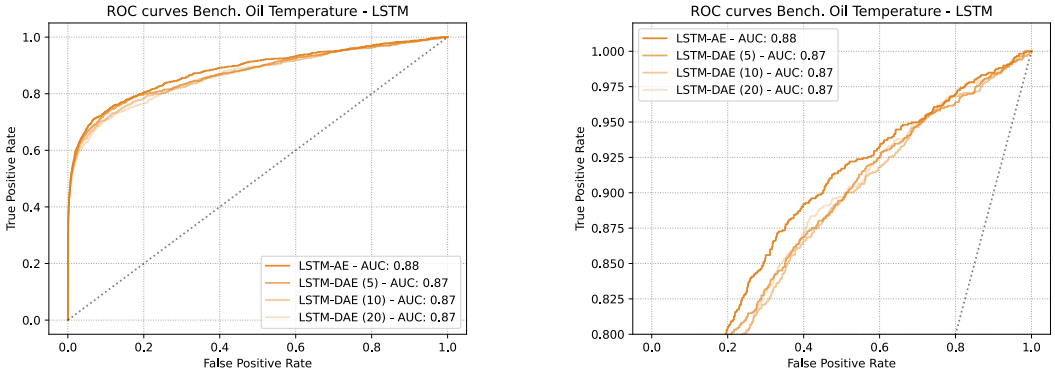
	Variance		
	0.8	**0.9**	**0.95**
PCA	.6805	0.7101	**0.7564**

Table 4.51: PCA results in terms of ROC-AUC scores for the *Gearbox Oil Temperature* benchmark.

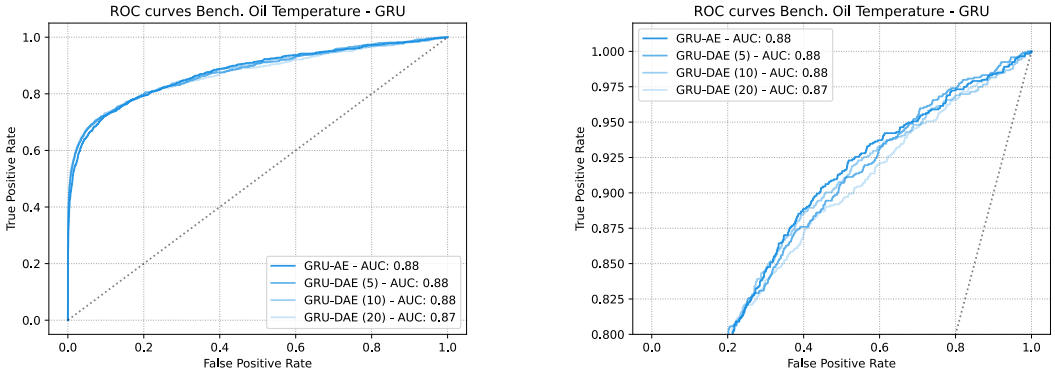
Figure 4.31 depicts the ROC curves for the most significant models that were utilized.



((a)) ROC curves for AE.



((b)) ROC curves for LSTM.



((c)) ROC curves for GRU.

Figure 4.31: ROC curves for all filters involve in training for AE and DAE models. Left: whole curve. Right: curve zoomed on high TPR.

Rotor RPM Benchmark

The findings for the *Rotor RPM* benchmark phase of anomaly identification are provided below. As demonstrated in the preceding section, it is evident that recurrent models can produce high-quality output. Similar to the preceding scenario, the score are just aligned as the ones from the validation phase. RNN models provide better results than the FNN ones.

Table 4.52, 4.53 and 4.54 show in that order, the results of FFN, LSTM and GRU. The tables show the ROC-AUC score values at varying hyperparameters of noise variance (in the autoencoder denoise problrm) and dropout probability.

		Dropout Rate	
		0%	**10%**
Noise	**NO**	**0.8456**	0.8332
	5%	0.8321	0.8428
	10%	0.8332	0.8375
	20%	0.811	0.8213

		Dropout Rate		
		0%	**10%**	**20%**
Noise	**NO**	0.8568	0.84	0.8849
	5%	0.8472	0.9017	**0.9096**
	10%	0.8535	0.8792	0.8792
	20%	0.8344	0.9005	0.849

Table 4.52: FNN AE results in terms of ROC-AUC scores for the *Rotor RPM* benchmark.

		Dropout Rate	
		0%	**10%**
Noise	**NO**	0.9303	0.9303
	5%	0.9301	0.9265
	10%	0.9207	0.9349
	20%	**0.9394**	0.9281

		Dropout Rate		
		0%	**10%**	**20%**
Noise	**NO**	**0.9714**	0.9533	0.9444
	5%	0.9662	0.9579	0.9447
	10%	0.9634	0.9498	0.9481
	20%	0.9574	0.9527	0.9485

Table 4.53: LSTM-AE results in terms of ROC-AUC scores for the *Rotor RPM* benchmark.

		Dropout Rate	
		0%	**10%**
Noise	**NO**	0.9335	0.9316
	5%	0.9316	0.9356
	10%	0.9307	0.9293
	20%	0.9377	**0.9382**

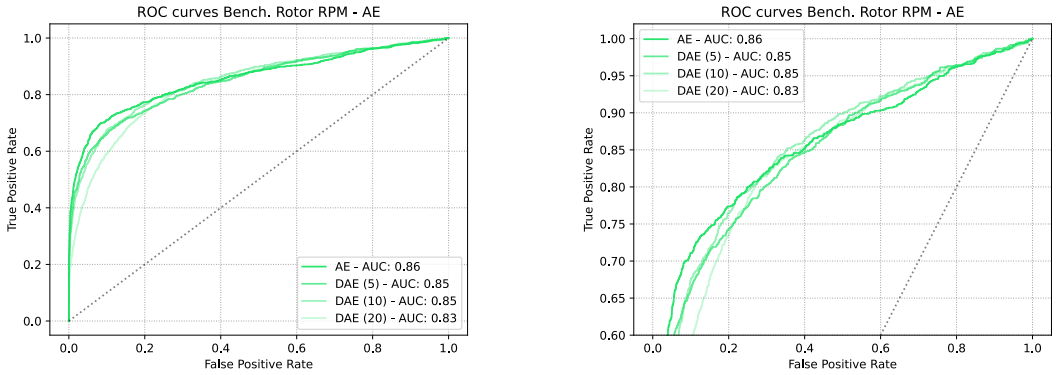
		Dropout Rate		
		0%	**10%**	**20%**
Noise	**NO**	0.9621	0.9583	0.9558
	5%	0.9604	0.9613	0.9554
	10%	**0.9677**	0.96	0.9551
	20%	0.9626	0.9555	0.9421

Table 4.54: GRU-AE results in terms of ROC-AUC scores for the *Rotor RPM* benchmark.

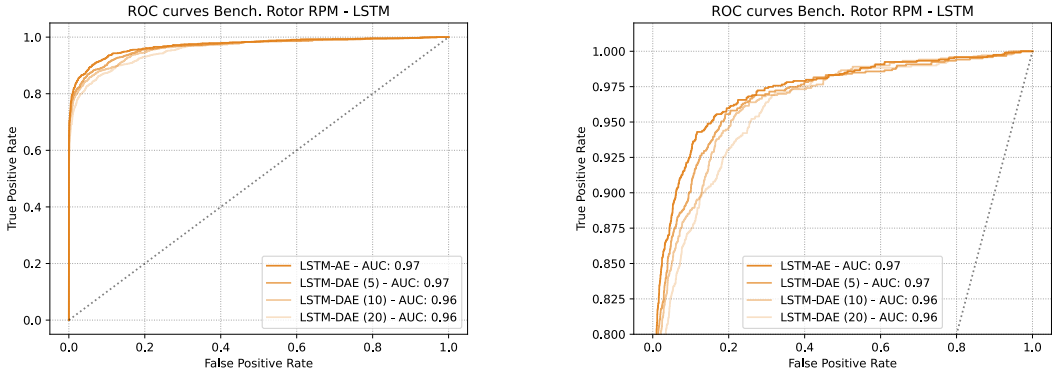
	Variance		
	0.8	**0.9**	**0.95**
PCA	.7242	0.7542	**0.7894**

Table 4.55: PCA results in terms of ROC-AUC scores for the *Rotor RPM* benchmark.

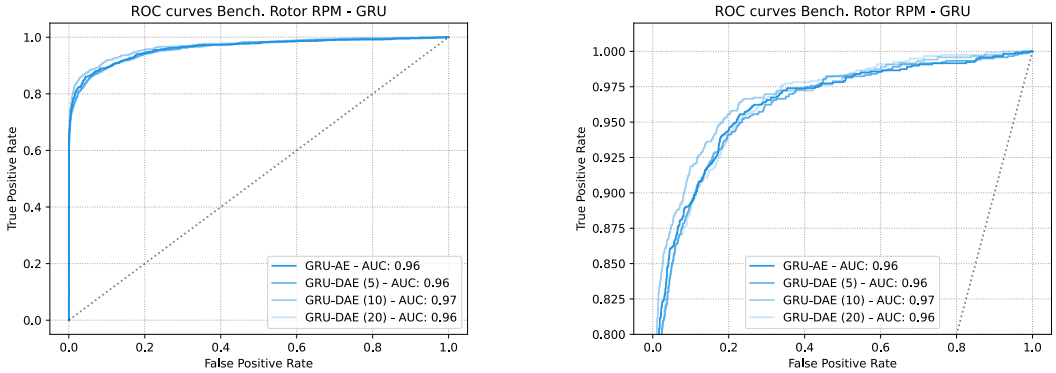
Again, some ROC curves for the most important models that were employed are depicted in Figure 4.32 that may be found below.



((a)) ROC curves for AE.



((b)) ROC curves for LSTM.



((c)) ROC curves for GRU.

Figure 4.32: ROC curves for all filters involve in training for AE and DAE models. Left: whole curve. Right: curve zoomed on high TPR.

4.4.6 Downtimes Detection Testing

During this phase of the downtime detection process, the group of turbines that were used for testing will be analyzed for the presence of anomalies, and an effort will be made to track the model results back to an interpretation of the observed phenomenon by virtue of the subsystems that make up the model.

In particular, the following downtimes experienced by the set of turbines that will be analyzed will be investigated. As was the case in the earlier example (concerning the training and validation turbine data), downtime moments have an extraordinarily varied character and have almost never occurred before in the history of a turbine. These behaviors, which are compounded, are found to occur within a group of anomalies on several quantities. The reason for this can be traced back to the fact that turbine subsystems are externally related to one another. However, the problem is further complicated by the fact that, once triggered, downtimes typically result in the activation of additional measures that are intended to suppress the phenomenon.

The studied downtimes are:

- **WTG-6**:
  - **2022/10/24**: *High gearbox bearing temperature* from 00:03:40 to 00:38:35.
  - **2022/10/24**: *High gearbox bearing temperature* from from 15:50:40 to 13:04:55.
  - **2022/10/24**: *High gearbox bearing temperature* from from 13:18:35 to 13:31:35.
  - **2022/10/24**: *High gearbox bearing temperature* from from 14:00:00 to 14:14:25.
  - **2022/10/24**: *High gearbox bearing temperature* from from 14:24:40 to 16:39:55.
- **WTG-7**:
  - **2021/11/02**: *High gearbox oil temperature* from 04:27:52 to 05:02:24.
  - **2021/11/02**: *High gearbox oil temperature* from 05:07:58 to 05:52:58.
  - **2021/11/02**: *High gearbox oil temperature* from 10:35:23 to 11:24:13.
  - **2021/12/02**: *High gearbox oil temperature* from 21:50:32 to 21:55:25.
  - **2021/12/02**: *High gearbox oil temperature* from 23:51:04 to 00:11:34.

High Gearbox Bearing Temperature

From the observation of the reconstruction error, it is evident how this phenomenon is be reported on the measure of *Gearbox Bearing Temperature* (Figure 4.33). There is a discernible rise in temperature during large part of the day, and this rise reaches its top value precisely at the times that are reported by the downtimes. A brake was applied in response to the sudden increase in temperature, which resulted in a precipitous reduction in temperature for both the oil and the bearings. This is another noteworthy event that occurred after the temperature rose.

The phenomenon also shows changes on the other parameters that were recorded, most notably on the parameters *Rotor RPM* and *Wind Speed* toward the end of the downtime period. In this scenario, the braking action brings an end to the rotation of the blades, which in turn causes the wind speed to increase in a manner that is disproportionate to the movement of the rotor.

The causes and effects of this phenomena are detailed in the following list. If the temperature of the bearing is too high, it is possible that the following subsystems and components will be involved:

- **Rotor**: That could cause the bearings to wear out faster than normal and cause them to overheat. In addition, if the blades are not pitched correctly, this might result in higher loading on the bearings, which in turn can cause the bearings to overheat.
- **Generator**: It is possible for the generator to produce an increased accumulation of heat in the bearings and other components, which can lead to the generator overheating if the generator is not adequately cooled or ventilated.

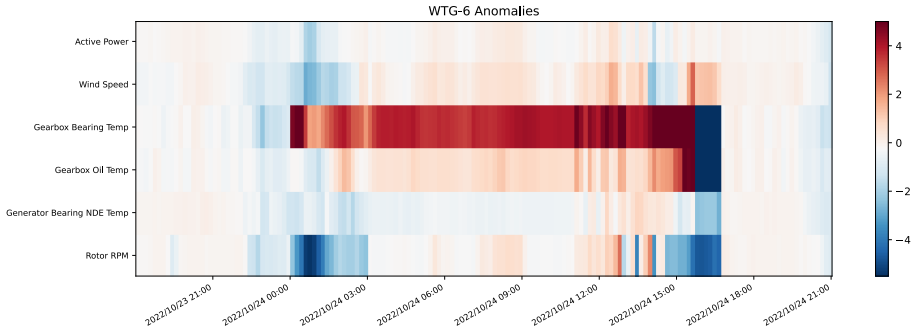


Figure 4.33: Reconstruction errors relative to the most significant features for the downtime *High Gearbox Bearing Temperature* registered on *2022/10/24* from *00:03:40* to *16:39:55*. Model used: GRU-AE with batch size 6, hidden space dimension of 64.

- **Lubrication system**: It is essential to have a good lubrication system in order to keep the bearings at the correct temperature and minimize wear. If the lubrication system is not maintained properly, it may cause reduced oil flow or oil degradation, both of which can lead to bearings that get too hot.
- **Cooling system**: The cooling system is in charge of dissipating the heat generated by a number of different components, including the bearings. If the cooling system is not operating as it should, the cooling capacity may be diminished, which may lead to an increase in the amount of heat that builds up in the bearings.

In the Tables 4.56, 4.57, 4.58 the results for the different models in the determination of error statuses and stopping moments are shown.

		Dropout Rate		
		0%	**10%**	**20%**
Noise	**NO**	0.9189	0.9031	0.9071
	5%	**0.9233**	0.9107	0.8975
	10%	0.9136	0.9182	0.8882
	20%	0.9117	0.9042	0.8805

Table 4.56: FNN AE results in terms of ROC-AUC scores for the 10 days surrounding the *High Gearbox Oil Temperature* downtime.

		Dropout Rate		
		0%	**10%**	**20%**
Noise	**NO**	0.9179	0.9095	0.957
	5%	0.9215	0.9244	0.9214
	10%	0.9227	0.9342	**0.9449**
	20%	0.9134	0.9264	0.9276

Table 4.57: LSTM-AE results in terms of ROC-AUC scores for the 10 days surrounding the *High Gearbox Oil Temperature* downtime.

		Dropout Rate		
		0%	**10%**	**20%**
Noise	**NO**	0.924	0.9112	0.9286
	5%	0.9468	0.8932	0.9026
	10%	0.9368	0.9258	0.9238
	20%	0.9195	**0.9339**	0.9039

Table 4.58: GRU-AE results in terms of ROC-AUC scores for the 10 days surrounding the *High Gearbox Oil Temperature* downtime.

High Gearbox Bearing Oil Temperature

From the observation of the reconstruction error, it is evident how this phenomenon is reported on the bearing oil temperature measurement 4.34. There is a noticeable increase in the temperature temperature during downtime reporting. In fact, between *4:00* and *6:00* a series of relevant bands are noted a high temperature encountered. The same is experienced between *10:00* and *11:00*. this phenomenon is associated with a prolonged rotor speed reduction effect probably due to the startup of an actuator for the brake procedure.

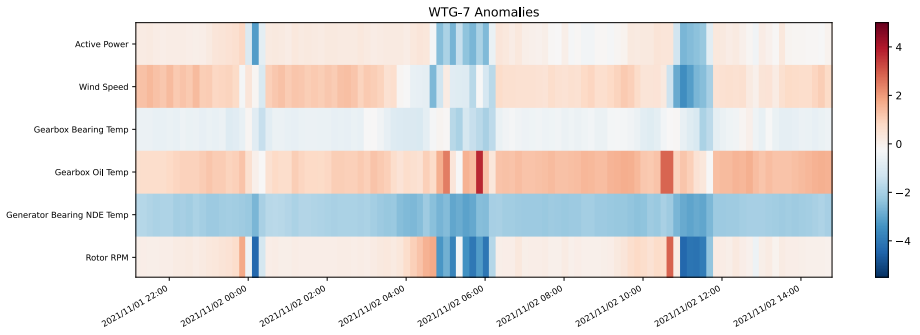


Figure 4.34: Reconstruction errors relative to the most significant features for the downtime *High Gearbox Bearing Oil Temperature* registered on *2021/11/02* from *04:27:52* to *11:24:13*. Model used: GRU-AE with batch size 6, hidden space dimension of 64.

One month after the previous event, another exceeding of critical bearing oil temperature thresholds is recorded. This time, however, the duration of the observation is different. In fact, it can be found a period of prolonged abnormality even after the triggering of the downtime (Figure 4.35).

When the oil temperature in the gearbox is too high, it can lead to a number of issues, including a reduction in the gearbox's overall efficiency and the possibility that some of its components will be damaged. In certain circumstances, the control system of the WTG may be programmed to react to high temperatures by slowing the rotor's rotational speed. This is done with the intention of saving the system from suffering additional damage.

The precise character of this response will be determined by the particular design of the control system and the WTG itself, respectively. This hypothesis, that the response is a countermeasure that has been programmed into the PLC, can be observed in the sudden nature of this reduction.

In the next Tables 4.59, 4.60, 4.61 the results for the different models in the determination of error statuses and stopping moments are shown.

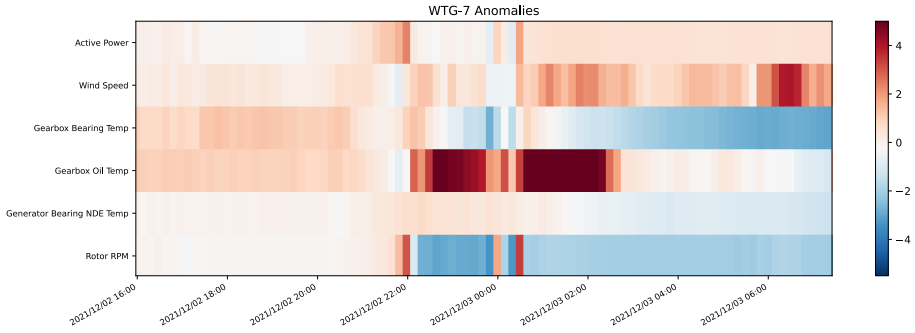


Figure 4.35: Reconstruction errors relative to the most significant features for the downtime *High Gearbox Bearing Oil Temperature* registered on *2021/12/02* from *04:27:52* to *00:11:34*. Model used: GRU-AE with batch size 6, hidden space dimension of 64.

		Dropout Rate		
		0%	**10%**	**20%**
Noise	**NO**	0.92	**0.9324**	0.9245
	5%	0.9229	0.9228	0.9203
	10%	0.9231	0.9338	0.9338
	20%	0.9195	0.9315	0.9238

Table 4.59: FNN AE results in terms of ROC-AUC scores for the 10 days surrounding both periods for the *High Gearbox Bearing Oil Temperature* downtime.

		Dropout Rate		
		0%	**10%**	**20%**
Noise	**NO**	0.93	0.9146	0.9226
	5%	0.9285	0.9274	0.9444
	10%	0.9239	0.933	**0.9429**
	20%	0.9256	0.9269	0.9225

Table 4.60: LSTM-AE results in terms of ROC-AUC scores for the 10 days surrounding both periods for the *High Gearbox Bearing Oil Temperature* downtime.

		Dropout Rate		
		0%	**10%**	**20%**
Noise	**NO**	0.9319	0.9233	0.9223
	5%	0.9327	**0.9352**	0.9371
	10%	0.9286	0.9262	0.9124
	20%	0.9261	0.9296	0.9196

Table 4.61: GRU-AE results in terms of ROC-AUC scores for the 10 days surrounding both periods for the *High Gearbox Bearing Oil Temperature* downtime.

4.5 Performance Analysis

The proposed framework is a valuable tool not only for anomaly detection of discrete events, but can also be employed for discovering unexpected or abnormal occurrences within a dataset. It is possible to use it to detect changes in behavior within a set of data, in addition to using it to identify certain sorts of anomalies, such as outliers or errors, which it can be applied to detect.

For instance, if a dataset contains information about wind turbines, anomaly detection can be exploited to determine whether or not a turbine is undergoing unanticipated changes in its performance or output. This could be the result of a malfunction in the turbine or a change in the conditions of the surrounding environment, either of which would be classified as an abnormality. On the other hand, the algorithm might also be used to detect more subtle changes in behavior, such as when a turbine begins to perform differently from others in the same cluster of turbines. This could be due to variations in the operating or maintenance methods, which might not be immediately obvious if an anomaly detection algorithm is not used.

Significant insights on the behavior of the wind turbines can be acquired by using such an algorithm, which will allow us to improve their overall performance and reduce the amount of time they are offline. In general, the application of anomaly detection algorithms is a strong tool that can be used to discover and handle any issues that may exist within datasets. These potential issues may include changes in behavior over the course of time.

When the reconstruction error is noticed, a measurement that refers to the gap between the actual value and the expected value can be derived. When it comes to wind turbines, there are a number of subsystems that contribute to the overall performance of the turbine. Some examples of these subsystems include the rotor, gearbox, generator, and control system. The performance of every individual subsystem can be evaluated using its own unique set of metrics, such as the rotor speed, the temperature of the gearbox, and the output of the generator.

When the values of these measures diverge from what is typical or expected, the newly created

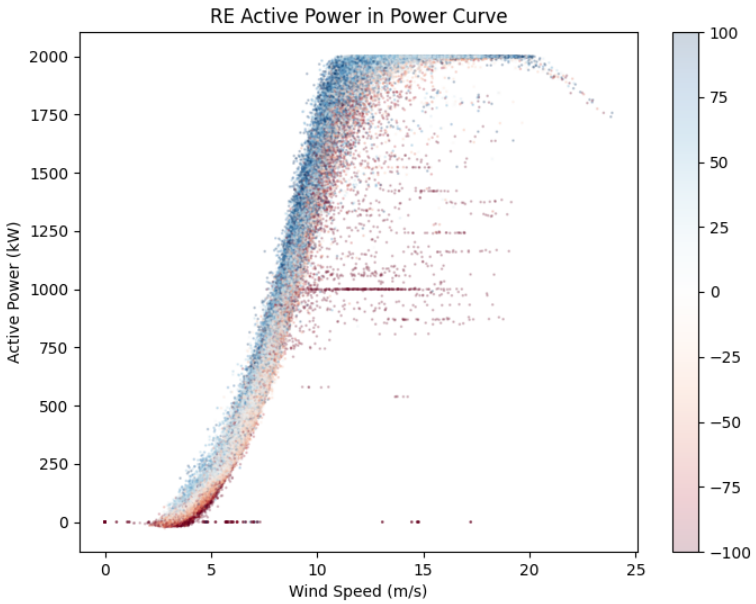


Figure 4.36: Power curve depicting the unscaled RE of the *Active Power* measure. In the image, which only display timesteps with non-error statuses, can be easily notice how the RE of this measure can be used to detected performance. In the image the RE is visualized with negative sign, making high performance appear in blue when an actual overpower is measured above the expected one. Since training data has a performance mean around 0.8, value above this threshold appear positively related.

method can be utilized to determine when this deviation has occurred. It is possible to spot problems and take remedial action before they become more serious if reconstruction error of each specific measure that is related to the subsystems of the wind turbine is monitored. This has the potential to assist us in enhancing the dependability and effectiveness of wind turbines, which will ultimately result in improved energy output and decreased downtime.

In order for the algorithm to function properly, the training dataset must be representative of the system being monitored. This indicates that the design and operating circumstances of the wind turbines used for training should be as similar as possible. If the training dataset includes wind turbines with significantly diverse designs or operating conditions, the model may not converge or be unable to detect unexpected behavior with precision. Due to the fact that the model was trained on a dataset that does not precisely represent the monitored system, it may not be able to reliably anticipate the behavior of the system under varied operating conditions.

The algorithm will be modified in this section of the study so that it can view macroscopically the set of test turbines that was used in the G90 trials. In point of fact, evaluations of the behavior of various subsystems can be carried out by observing the errors that occur during reconstruction. Under the framework of the investigation and upkeep of these subsystems, this observation might be interpreted as a use case of the algorithm.

The first indicator for obtaining macroscopic information on wind farm performance comes from observing the RE on the *Active Power* measure. It can be seen in the Figure 4.36 how this RE is particularly effective in observing the performance of operating systems. After all, performance filters allowed the study of ideal turbine behavior where average performance was considered to be around 80%. If AEs did not bring problems in the correlation of quantities, *Active Power* and *Wind Speed* are the most useful measures that implicitly represent performance. In the figure, where the error states have been removed, which otherwise would have made it difficult to observe in the part of the graph at the cut-out speed, this relationship is evident. It is worth noting that in the graph RE is calculated with a negative sign. Since it is normally considered as $pred - act$. The graph tries to link a positive RE with an overproduction of energy, so it was preferable to use $act - pred$. That way the expectation, being smaller than the actual production, would give a positive sign.

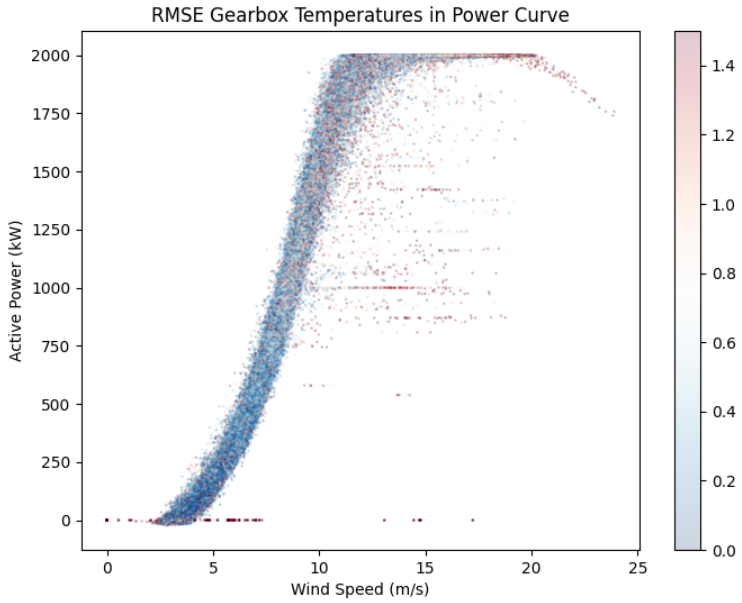


Figure 4.37: Power curve depicting the unscaled RMSE for the bearing temperature measures. In the image only timesteps with non-error statuses are displayed.

Nevertheless, when looking at other RE metrics, whether or not there is a correlation or connectivity with the power curve is not as easily discernible as it is when looking at the power

curve itself. This point is demonstrated in the figure that can be found in Figure 4.37. In this particular instance, the root mean square error (RMSE) between gearbox temperatures does not return the same result as what was demonstrated in the previous example. In point of fact, significant reconstruction errors are more common towards the top of the curve, and they become more common as the *Active Power* grows. Nonetheless, the greatest amount of reconstruction error was seen for these quantities at practically all of the points that were found below the curve, in the region that was quite a ways below the high point density.

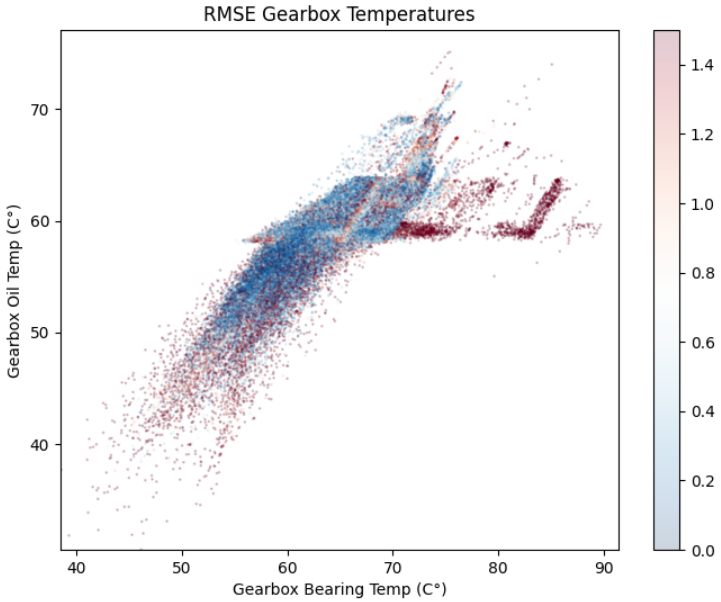


Figure 4.38: Scatter plot depicting the gearbox temperature measures, each point color represents the unscaled RMSE value. In the image only timesteps with non-error statuses are displayed.

This is a reliable result with what was said in the data filtering step about gearbox temperatures. The desired effect with these filters was to limit the difference in absolute value between the temperatures to within about 10 C°. This measure was chosen as a compromise to avoid including malfunctions but at the same time avoid over-cleaning the dataset, which would have led to substantial reductions in the number of data. This RMSE measurement on gearbox temperatures can be related more to the scatter plot of temperatures. One can esservare in the Figure 4.38 how this magnitude isolated the most distant points from the bisector.

According to what was just stated, the following table presents the active power RE measurements that were taken on average for each turbine that was part of the test set during the various months of the year. The data presented in the Table 4.62 are derived from time sequences that have had any erroneous periods eliminated.

The table presents the reconstruction error forthe WTGs, where the mean *Active Power* has been used for each month as the metric of interest. While this approach may seem reductive, it is still significant for analysis purposes. Additional metrics such as standard deviation, skewness, and kurtosis can also be calculated for the obtained distributions, providing further insights into the data. However, to avoid redundancy, only the mean has been reported in this table. When analyzing the efficacy of the reconstruction model, the reconstruction error is an essential measure to take into consideration. This is something that should be brought to your attention. It gives an indication of the degree to which the projected values and the actual values differ from one another.

Although it is possible to deepen the analysis on all the turbines and different periods, below is the case study of *WTG-6* turbine in September. This turbine in this time period has the lowest RE in the *Active Power* measure. The measurements reported in the table refer to the RE

RE Mean	**Devices**				
Month	WTG-5	WTG-6	WTG-7	WTG-8	WTG-9
Jan	-20.68	-5.07	-12.2	9.4	2.48
Feb	-20.41	0.66	-1.77	11.79	5.25
Mar	-21.17	-3.13	-4.62	4.19	6.85
Apr	-12.09	6.69	2.05	12.68	7.32
May	-28.87	-5.47	-8.23	8.05	4.37
Jun	-33.86	-12.76	-18.76	1.34	2.58
Jul	-45.34	-13.35	-8.07	7.44	-4.25
Aug	-37.8	-38.11	-12.57	-2.36	-6.84
Sep	-26.96	-59.86	-1.48	-2.47	-5.32
Oct	-33.77	-24.71	-10.08	-8.46	1.39
Nov	-18.56	12.65	5.56	15.94	17.93
Dec	-10.4	10.67	4.75	14.5	7.26

Table 4.62: Unscaled RE of the measured *Active Power* for the test WTGs during the year. All measures refer to the produced power and are expressed in kW.

measured always $act - pred$, moreover in the case of the table they were then scaled back to the original size therefore the measurements are in kW.

Despite the measurements tend to be lower for *WTG-5*, observations on *WTG-6* show a higher level of deviation than the average case of the latter. This turbine was selected to be monitored since, in the previous scenario, one could still be able to see a phenomenon of low productivity and failure, but a longer time period would be required to justify it.

It is also generally true, as can be seen from the observation of the table, that turbines appear to have a lower production of energy during the summer. This is understandable due to the fact that a WTG's primary function is to transform the kinetic energy of wind into electrical energy. The amount of energy that can be generated by a WTG is contingent upon a number of variables, including as the wind speed, the air density, the size of the turbine, and its level of efficiency. Alterations in the pressure and temperature of the atmosphere tend to bring about shifts in the wind patterns that prevail during the summer months. The average wind speed during the summer months is significantly lower in many areas than it is during the other seasons. Because of this, WTGs produce less power. In addition, summer temperatures can reduce the efficiency of WTG components like the rotor blades and generator, reducing energy production. Heat can weaken and expand materials, decreasing aerodynamic performance and rotor blade resistance.

This effect, though diminished, is still evident in the results. To reduce it further, the data might include an indication of the symbolic on the measurement time to induce the AEs in a different representation at these points. This would lessen the effect. In addition to symbolic measurements of system state and environment, air density should be measured. Using air temperature readings would make sense, however during the data selection process, it was found that these measurements are already confined and have a high link to other turbine temperatures. These measurements are ignored for the problem.

WTG-6 Performance Analysis

In comparing the performance among different turbines, *WTG-6* was initially considered. This turbine appears in the list of turbines on which downtime is present, in addition, the objective of this study is to determine how the occurrence in question may be connected to differences in the manner in which reconstruction errors are distributed.

Through the use of violin plots, which allow us to visualize the distribution of reconstruction error during the year, it is possible to examine this behavior on the amount of active power. This RE is generated by taking into account the difference in magnitude between what was observed and what was expected, and by doing so, attempting to highlight in the positive or negative variance a direct representation of the magnitude that was either over- or under-reported.

In Figure 4.39, for the size of *Active Power*, there is a fairly regular behavior throughout the year with a definite deviation during the month of September. In this month the RE register a actual energy production lowest than the expectation.

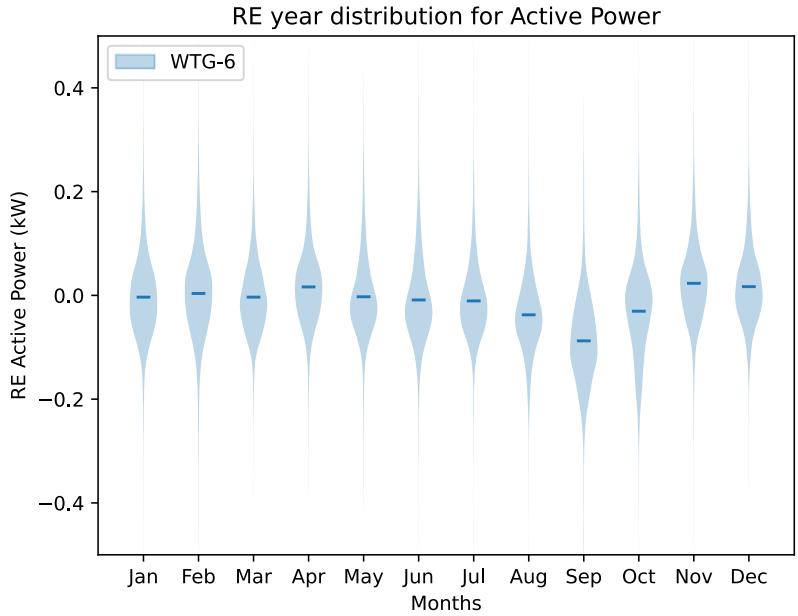


Figure 4.39: *Active Power* RE violin plots for each month.

The month in question is highlighted in the downtime event on the magnitude of *Gearbox Bearing Temperature*, and as a result, the following violin graph (Figure 4.40) will display precisely this measurement.

It is evident from the graph that the observed measure is significantly deviated from previous ones. In addition to this measure, others are also given below (Figure 4.41 and Figure 4.42), for the magnitudes of *Gearbox Oil Temperature* and *Rotor RPM*. These were the other measures involved in the observation on the downtimes that occurred.

The latter metrics, despite having variances that cannot be ignored, are nevertheless rather insignificant in comparison to the prior ones in terms of size. Last but not least, the curve of the gearbox bearing temperature is displayed in Figure 4.43.

In the graph shown, they are clearly distinguishable:

- A period of lasting discontinuity, in which the temperature undergoes a significant rise for about a week.
- A peak, at the downtime recorded subsequent to the period when the temperature is high.

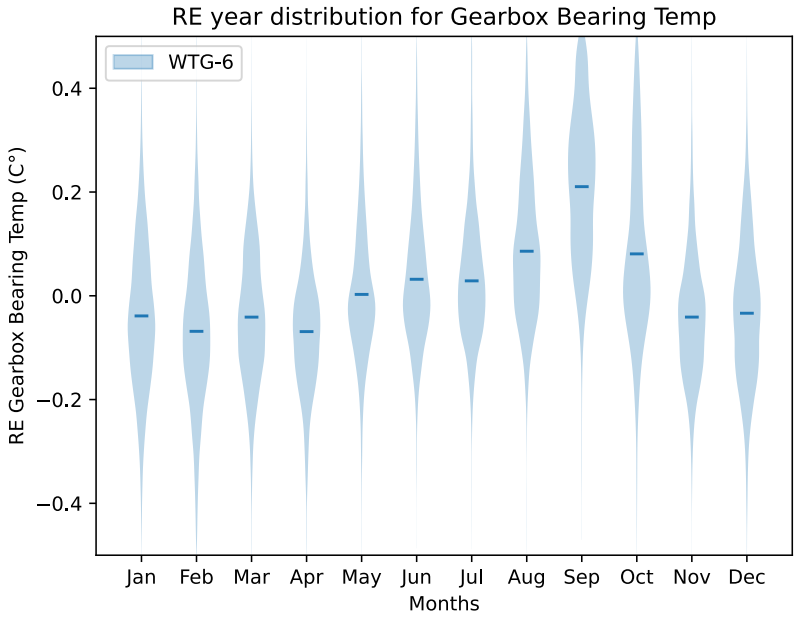


Figure 4.40: *Gearbox Bearing Temperature* RE violin plots for each month.

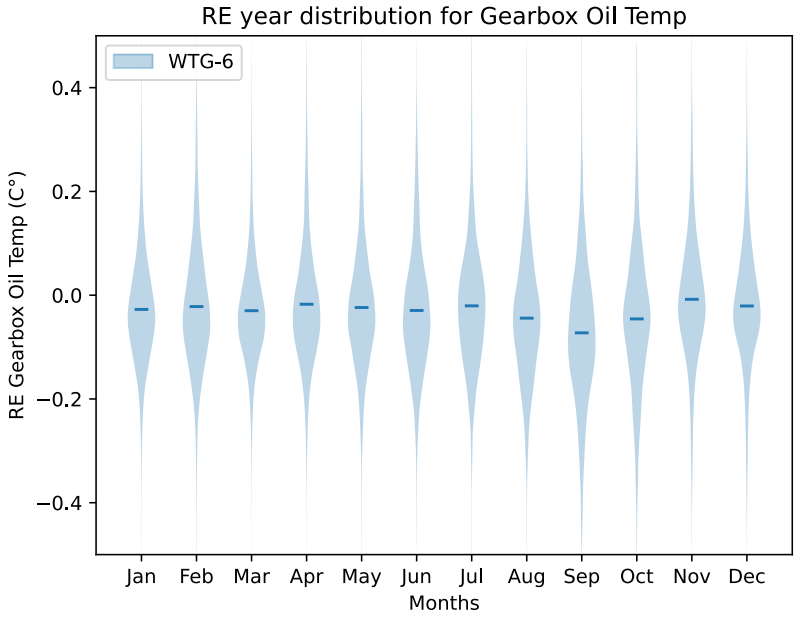


Figure 4.41: *Gearbox Oil Temperature* RE violin plots for each month.

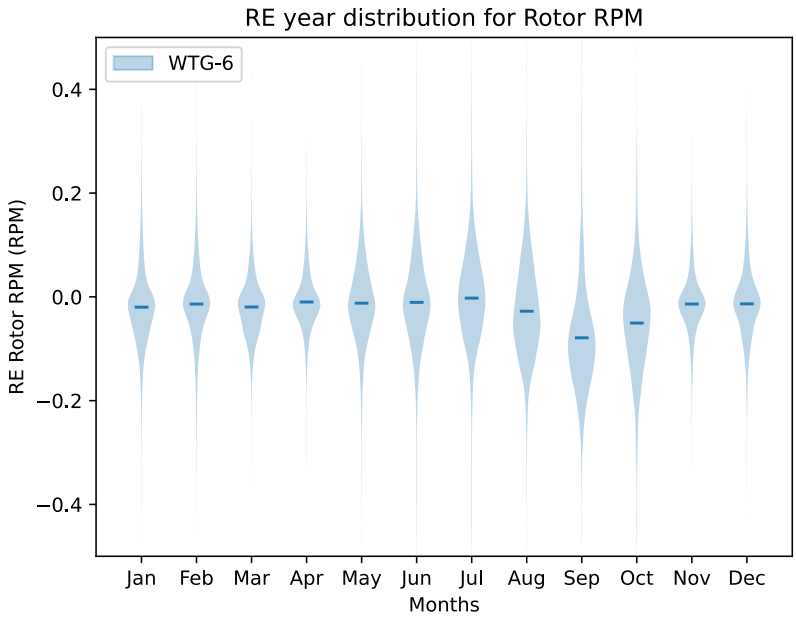


Figure 4.42: *Rotor RPM* RE violin plots for each month.

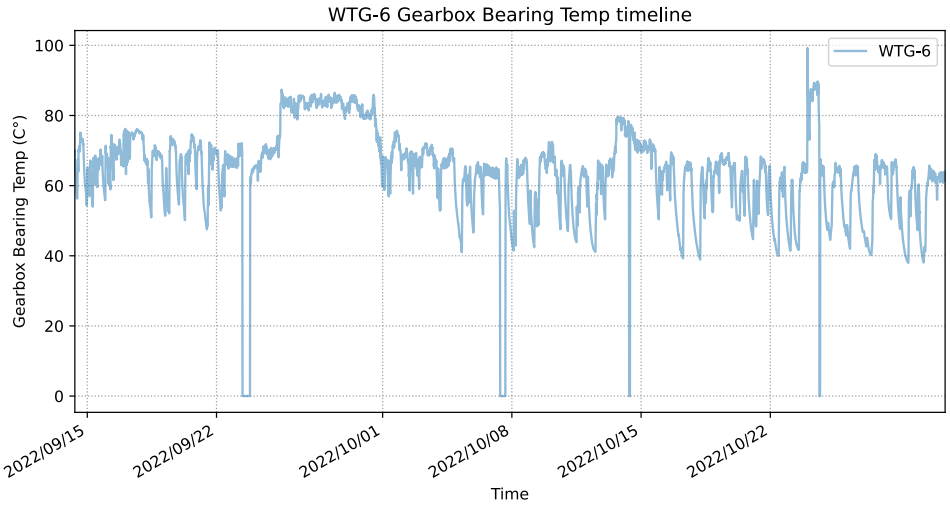


Figure 4.43: *Gearbox Bearing Temperature* plot from august and october for *WTG-6*.

4.6 Gamesa G87

- Power:
  - Rated power: 2.00 MW.
  - Cut-in wind speed: 3.5 m/s.
  - Cut-out wind speed: 25 m/s.
  - Survival wind speed: 50 m/s
- Rotor:
  - Blade number: 3.
  - Diameter: 87 m.
  - Area: 5,945 m^2.
- Gearbox:
  - Type: spur/planetary.
  - Gear stages: 3.
  - Transmission Ratio: 0.101.
  - Brand: Echesa (Gamesa Group)/Hansen/Bosch Rexroth/Winergy.
- Generator:
  - Type: Doubly-fed Asynchronous.
  - Brand: Cantarey.
  - Voltage: 690 V.
  - Main frequency: 50 Hz.
  - Speed max: 1900 U/min.
- Weight:
  - Single Blade: 6.2 t.
  - Hub: 18.6 t.
  - Rotor: 37.0 t.
  - Nacelle: 107.0 t.
  - Tower: 242.0 t.
  - Total: 386.0 t.

4.6.1 G87 Cross Testing

At this point in the trial, a cross-experiment will be undertaken, during which models that were previously trained on G90 turbine types will be utilized with G87 turbine types. In order to establish some sort of basis for comparison, these models will be confounded with models that are identical to them (in terms of their hyperparameters) but are towed directly by G87s. The experiment, which at first glance seems unfair (given that the models that have been towed up until this point have never come into contact with this type of turbine), turns out to be motivated by the fact that the models that are trained on G87s will have a significantly smaller dataset available to them. The question that arises next is whether it is more advantageous to adopt a technique in which AE models are built using fewer data from the target type turbine, or whether it is more advantageous to choose pretrained models using extraneous datasets with more data.

As the only factor that varies the data that were looked at is the type of turbine that was investigated, it is quite reasonable to raise this question. The distinctions between these models can be boiled down to relatively insignificant discrepancies in terms of geometric and mechanical

characteristics, however there are some aspects of the wind farm in question that are shared. Since these models are all situated in the same region of the world, the environmental factors are uniform across all of them.

The experiment involves using the models firstly developed for dataset G90 on data from cluster G87 to detect anomalies in the operation of the wind turbines. After taking into account the results that were produced with the models that were trained using the G90 dataset, those models are then tested on a fresh dataset that comes from the G87 cluster to determine how well they perform on new data. The dataset obtained from cluster G87 comprises sensor measurements that are quite comparable to those obtained from cluster G90; however, the operating conditions and physical features of the turbines in cluster G87 may be different.

Further models are trained directly on the G87 dataset utilizing the same evaluation criteria and algorithms in order to conduct a head-to-head comparison of the performance of the first models on the G87 dataset. Following this step, the performance of the various models is evaluated based on their respective receiver operating characteristic (ROC) curves.

The outcomes of this experiment have the potential to shed light on the generalizability of the models trained on the G90 to fresh datasets as well as various types of wind turbine clusters. If the first models perform well on the new dataset, this suggests that they are able to detect anomalies in a variety of wind turbine clusters successfully, which is an essential need for anomaly detection in industrial systems. On the other hand, if the models do not perform well on the G87 dataset, this indicates that the models may require additional training or optimization in order to increase their generalizability.

4.6.2 Data Selection

As in previous experiments, the dataset considered will be divided into two cluters:

- **High-performance (Training/Validation) turbines**: These turbines will be used to train effectively the models used for anomaly detection. However, this time the experiments will consider as the anomaly detection scenario, only the next cluster. This dataset will not be tested with anomaly detection, however, time periods related to the detection phases will still be removed. This cluster is made of 2 turbines, namely: *WTG-10* and *WTG-11*.

- **Other (Testing) turbines**: During the testing phase, these turbines will be utilized solely for the purpose of testing the performance of the algorithm on data derived from unobserved turbines (in the training phase). The models will be able to be evaluated using new data that is somewhat distinct from the data that was used in the prior tests thanks to this test. This cluster is made of 3 turbines, namely: *WTG-12*, *WTG-13* and *WTG-14*.

As was mentioned earlier, the number of turbines that will be taken into consideration this time will be cut down. Ensuring that there is at least a level playing field when compared to the challenger models that were trained earlier. This time, the potential of mixing data from a large number of turbines in the training phase will be decreased; in fact, there will be just two training devices, which increases the danger of not fully obtaining the level of generalization that is actually wanted.

Once again, data will be evaluated over an entire year. The produced splits are:

- **Training split**: For the first 3 months of the four-month period, after the filtering process described next.

- **Validation split**: Random samples from the previous set.

- **Testing split**: Consists of the last month in the four-month period. Not used in this experiment.

Measure	Lower Bound	Upper Bound	Other Conditions
Active Power	-50kW	2300kW	
Wind Speed	3.5m/s	25m/s	
Performance (G87)	40%	200%	
ABS Temp. Difference		12°	
Status			Not error status
Perf. (G87) Cent. MA 12	80%	150%	
Limitations			Not a limitation

Table 4.63: Types of filters used in the filtering phase for the G87 dataset. The table shows the list of quantities used and the appropriate filter conditions. All data less than the lower bound or greater than the upper bound were removed. Since *Status* is a symbolic data type the *Error Status* is removed.

4.6.3 Data Filtering

After the extraction of data for each individual turbine, the data was filtered. The filters used are described in Table 4.63.

Given the small number of data available and the different shape of the G87 power curve, filters on performance were considered with reduced severity. The defined result therefore should be a dataset with little more than 50% of the previous one.

4.6.4 Sequence Processing

At the end of the preprocessing phase, the sequences obtained are about two-thirds of those available in the G90 dataset. Table 4.64 shows more information about the splits.

WTG	N° Samples (Seqs.)	% Samples (Seqs.)	% Valid Timesteps
WTG-10	6812/39304	17.3%	36.74%
WTG-11	14442/39304	36.74%	51.26%

Table 4.64: Data samples (in sequence format) after filtering and preprocessing for G87 dataset for cross test. The fact that the number of valid sequences does not equal the number of valid timestamps resides in the fact that only sequences solely consisting of valid data are usable in the training phase.

4.6.5 Models Processing

Models are chosen based on the results of previous experiments. The models chosen for this experiment have the following characteristics:

- **Batch size**: 12.
- **Dropout**: 0% and 10%.
- **Gaussian Noise**: None or 10%.
- **Latent dimension**: 48.
- **Network type**: FNN, LSTM and GRU.

4.6.6 Anomaly Detection in Validation and Testing Datasets

Leaving out the description on training and validation of models and calculation of residuals. At this stage, attention is being focused on the testing phase of the process. This stage will be focused entirely on the application of the event discrimination practice.

In the Tables 4.65, 4.66 and 4.67 results are shown for FNN, LSTM and GRU model in anomaly detection for training filters.

		Dropout Rate	
		0%	**10%**
Noise	**NO**	**0.9016**	0.9012
	10%	0.9008	0.8958

		Dropout Rate	
		0%	**10%**
Noise	**NO**	**0.8969**	0.8716
	10%	0.8847	0.8631

Table 4.65: FNN AE results in terms of ROC-AUC scores for for discrimination based on the same filters used in training. Left: G87 models. Right: G90 models.

		Dropout Rate	
		0%	**10%**
Noise	**NO**	0.9073	0.9205
	10%	0.9146	**0.9224**

		Dropout Rate	
		0%	**10%**
Noise	**NO**	0.8995	0.9100
	10%	0.8871	**0.9135**

Table 4.66: LSTM-AE results in terms of ROC-AUC scores for for discrimination based on the same filters used in training. Left: G87 models. Right: G90 models.

		Dropout Rate	
		0%	**10%**
Noise	**NO**	0.9051	0.9173
	10%	0.8904	**0.9288**

		Dropout Rate	
		0%	**10%**
Noise	**NO**	0.8704	0.9031
	10%	0.8901	**0.9118**

Table 4.67: GRU-AE results in terms of ROC-AUC scores for for discrimination based on the same filters used in training. Left: G87 models. Right: G90 models.

It can be seen from the results that the values obtained from the models towed on the G87s are very similar to those for the G89 models and tested in the corresponding datasets. The values that were obtained from the FNN models, on the other hand, were much lower. This is most likely because the network had a reduced capacity to assess the dataset that was being analyzed due to the fact that it had a smaller number of training parameters. High results are produced by the RNN models, and they are pretty equivalent to those obtained on the prior dataset.

In terms of the outcome of the cross testing, the G90 models have generally lower performances, but the results are satisfactory taking into consideration the fact that the data reflect various kinds of turbines. The FNNs have a poorer performance in this regard, but the recurrent models, particularly those with a higher degree of regularization, are able to drastically diminish the effect.

In the Tables 4.68, 4.69 and 4.70 results are shown for FNN, LSTM and GRU model in anomaly detection for *Inactive* and *Error* statuses.

		Dropout Rate	
		0%	**10%**
Noise	**NO**	0.9316	**0.9324**
	10%	0.9188	0.9058

		Dropout Rate	
		0%	**10%**
Noise	**NO**	**0.9231**	0.9208
	10%	0.9206	0.8958

Table 4.68: FNN AE results in terms of ROC-AUC scores for discrimination using only *Error* and *Inactive* statuses. Left: G87 models. Right: G90 models.

		Dropout Rate	
		0%	**10%**
Noise	**NO**	0.9273	0.9205
	10%	0.9346	**0.9456**

		Dropout Rate	
		0%	**10%**
Noise	**NO**	0.9273	0.9105
	10%	0.9346	**0.9360**

Table 4.69: LSTM-AE results in terms of ROC-AUC scores for discrimination using only *Error* and *Inactive* statuses. Left: G87 models. Right: G90 models.

		Dropout Rate	
		0%	**10%**
Noise	**NO**	0.9235	0.9273
	10%	0.9207	**0.9478**

		Dropout Rate	
		0%	**10%**
Noise	**NO**	0.9166	0.914
	10%	0.9263	**0.9380**

Table 4.70: GRU-AE results in terms of ROC-AUC scores for discrimination using only *Error* and *Inactive* statuses. Left: G87 models. Right: G90 models.

The findings of the most recent experiment add legitimacy to the statements made above. However, the AUC scores pick up on more irregular scores, most likely as a result of the varied characteristics of the datasets that were taken into consideration. In point of fact, it is not possible to rule out the possibility that different models of the G87 have distinct sensors and control features, as well as varied ways of reporting issues linked to failure and error status.

Chapter 5

Conclusions

The thesis emphasizes the significance of anomaly detection algorithms in the process of determining the performance of turbines and the subsystems that comprise them. The obtained results highlight the fact that FNN and RNN techniques are preferable to linear algorithms when it comes to reconstructing multivariate time series data.

The proposed solution comprises the development of a variety of AE models that are based on these approaches. In addition to this, the regularization procedures of DAE and VAE are the primary emphasis of the thesis since they offered significant benefits in comparison to more conventional models. The research demonstrates that multivariate time series data, particularly those that are derived from sensors and 10-minute data, are frequently difficult to understand and require the use of advanced algorithms for reconstruction and interpretation. This thesis focuses on the data that are derived from 10-minute data. When working with these types of data, linear algorithms are not always sufficient; rather, more complex approaches are required to handle the situation. The AE models that were proposed were able to capture the underlying patterns and relationships that were present in the data because they were based on approaches such as FNN and RNN. In comparison to linear algorithms, the end consequence of their work was an increase in the accuracy of the reconstructed image. This is of the utmost significance for systems such as turbines, where even tiny differences in performance can result in severe problems and downtime if the issue is not handled immediately. The validity and reliability of the algorithms were established through the execution of different types of tests, the results of which demonstrated the viability of the solution that was proposed. Other than these tests, the work offers a way of use for these algorithms within real-world contexts in which the observation of quantities often requires numerous techniques for determining outliers. In addition, the presence of many time series that were distinguished by downtime and alarms made it possible for the algorithms to be qualitatively evaluated throughout the process of composing failure triggers.

In general, this thesis contributes to the field of anomaly detection and performance evaluation by presenting a holistic method that applies to a wide variety of fields and systems, including the energy industry.

5.1 Future Works

In this thesis, an unsupervised strategy was used due to the absence of many strongly marked anomalous phenomena. In this thesis, an unsupervised strategy was used due to the absence of many strongly marked anomalous phenomena. This was done because of the lack of supervision. Because of this, it became feasible to examine certain events indirectly rather than just by observing them directly. On the other hand, with access to a dataset that contains clearly delineated phenomena, a supervised method could be employed. Under the context of such an approach, multiple categories of anomalies would be investigated (each of which would be given a specific class label), and during testing, a symbolic indicator would be produced without the necessity of resorting to an explicit computation of an aggregation of the differences. On the other hand, with

access to a dataset that contains delineated phenomena, a supervised method could be employed. Under the context of such an approach, multiple categories of anomalies would be investigated (each of which would be given a specific class label), and during testing, a symbolic indicator would be produced without the necessity of resorting to an explicit computation of an aggregation of the differences. In order to make the reality described by the AEs more sophisticated and accurate, symbolic information linked to quantities that were measured and extracted by various algorithms may subsequently be added to the data that was fed to the encoders. These symbolic indicators could have something to do with different indications regarding the consequences of WTG models, or they could provide information about the context in which the observation was made.

After that, such a method could be trained with the help of anomalies produced by a generative model. In the event that these anomalies can be separated from one another with relative ease, it would be possible to devise a system that would be able to realistically integrate a limited number of anomalies with an uncontaminated time series. A technique of this kind might thus also be trained by employing anomalies produced by a generative model. If these anomalies can be separated from one another with relative ease, it would be possible to devise a system that would be able to integrate a limited number of anomalies with an uncontaminated time series. This would also help solve the dearth of recorded anomalies by allowing for the production of models that are more accurate in their identification of them.

One last point to consider is that the selection of the measures can be different depending on the type of dataset that was used. Because of the presence of data that were obtained using averages over extensive periods, the models that were investigated are restricted in their grasp of the intricacy of the phenomenon (10 minutes). By utilizing data that is characterized by higher frequencies (a few seconds or even less), it would be possible to analyze abnormalities and decreases in performance that is even caused by short transitory variations. Ensuring a level of comprehension of the event that is more precise with each passing day.

Bibliography

[1] Soteris A Kalogirou. "Artificial neural networks in renewable energy systems applications: a review". In: *Renewable and sustainable energy reviews* 5.4 (2001), pp. 373–401.

[2] Dennis YC Leung and Yuan Yang. "Wind energy development and its environmental impact: A review". In: *Renewable and sustainable energy reviews* 16.1 (2012), pp. 1031–1039.

[3] Maria Isabel Blanco. "The economics of wind energy". In: *Renewable and sustainable energy reviews* 13.6-7 (2009), pp. 1372–1382.

[4] Wei Qiao and Dingguo Lu. "A survey on wind turbine condition monitoring and fault diagnosis—Part I: Components and subsystems". In: *IEEE Transactions on Industrial Electronics* 62.10 (2015), pp. 6536–6545.

[5] Peter J Schubel and Richard J Crossley. "Wind turbine blade design". In: *Energies* 5.9 (2012), pp. 3425–3449.

[6] Gürdal Ertek and Lakshmi Kailas. "Analyzing a decade of wind turbine accident news with topic modeling". In: *Sustainability* 13.22 (2021), p. 12757.

[7] Kira Grogg. "Harvesting the wind: the physics of wind turbines". In: *Physics and Astronomy Comps Papers* 7 (2005).

[8] Albert Betz. *Introduction to the theory of flow machines*. Elsevier, 2014.

[9] M Lydia et al. "A comprehensive review on wind turbine power curve modeling techniques". In: *Renewable and Sustainable Energy Reviews* 30 (2014), pp. 452–460.

[10] James F Manwell, Jon G McGowan, and Anthony L Rogers. *Wind energy explained: theory, design and application*. John Wiley & Sons, 2010.

[11] Rosmaini Ahmad and Shahrul Kamaruddin. "An overview of time-based and condition-based maintenance in industrial application". In: *Computers & industrial engineering* 63.1 (2012), pp. 135–149.

[12] Andrew KS Jardine, Daming Lin, and Dragan Banjevic. "A review on machinery diagnostics and prognostics implementing condition-based maintenance". In: *Mechanical systems and signal processing* 20.7 (2006), pp. 1483–1510.

[13] Suprasad V Amari and Leland McLaughlin. "Optimal design of a condition-based maintenance model". In: *Annual Symposium Reliability and Maintainability, 2004-RAMS*. IEEE. 2004, pp. 528–533.

[14] David Parmenter. *Key performance indicators: developing, implementing, and using winning KPIs*. John Wiley & Sons, 2015.

[15] DJ Willis et al. "Wind energy research: State-of-the-art and future research directions". In: *Renewable Energy* 125 (2018), pp. 133–154.

[16] Elena Gonzalez et al. "Key performance indicators for wind farm operation and maintenance". In: *Energy Procedia* 137 (2017), pp. 559–570.

[17] Bryan Lim and Stefan Zohren. "Time-series forecasting with deep learning: a survey". In: *Philosophical Transactions of the Royal Society A* 379.2194 (2021), p. 20200209.

[18] Robert H Shumway, David S Stoffer, and David S Stoffer. *Time series analysis and its applications*. Vol. 3. Springer, 2000.

[19] Pedro Domingos. "A few useful things to know about machine learning". In: *Communications of the ACM* 55.10 (2012), pp. 78–87.

[20] Rui Xu and Donald Wunsch. "Survey of clustering algorithms". In: *IEEE Transactions on neural networks* 16.3 (2005), pp. 645–678.

[21] Jonathon Shlens. "A tutorial on principal component analysis". In: *arXiv preprint arXiv:1404.1100* (2014).

[22] Laurens Van der Maaten and Geoffrey Hinton. "Visualizing data using t-SNE." In: *Journal of machine learning research* 9.11 (2008).

[23] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. "Isolation forest". In: *2008 eighth ieee international conference on data mining*. IEEE. 2008, pp. 413–422.

[24] Athanasios Voulodimos et al. "Deep learning for computer vision: A brief review". In: *Computational intelligence and neuroscience* 2018 (2018).

[25] Jayashree Padmanabhan and Melvin Jose Johnson Premkumar. "Machine learning in automatic speech recognition: A survey". In: *IETE Technical Review* 32.4 (2015), pp. 240–251.

[26] Amirsina Torfi et al. "Natural language processing advancements by deep learning: A survey". In: *arXiv preprint arXiv:2003.01200* (2020).

[27] Sorin Grigorescu et al. "A survey of deep learning techniques for autonomous driving". In: *Journal of Field Robotics* 37.3 (2020), pp. 362–386.

[28] Yoshua Bengio. "Practical recommendations for gradient-based training of deep architectures". In: *Neural Networks: Tricks of the Trade: Second Edition* (2012), pp. 437–478.

[29] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[30] Bing Xu et al. "Empirical evaluation of rectified activations in convolutional network". In: *arXiv preprint arXiv:1505.00853* (2015).

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

[32] Quoc V Le, Navdeep Jaitly, and Geoffrey E Hinton. "A simple way to initialize recurrent networks of rectified linear units". In: *arXiv preprint arXiv:1504.00941* (2015).

[33] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. "On the difficulty of training recurrent neural networks". In: *International conference on machine learning*. Pmlr. 2013, pp. 1310–1318.

[34] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[35] *LSTMs Explained: A Complete, Technically Accurate, Conceptual Guide with Keras*. https://medium.com/analytics-vidhya/lstms-explained-a-complete-technically-accurate-conceptual-guide-with-keras-2a650327e8f2.

[36] Kyunghyun Cho et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *arXiv preprint arXiv:1406.1078* (2014).

[37] *What is the difference between LSTM and GRU?* https://www.nomidl.com/deep-learning/what-is-the-difference-between-lstm-and-gru/.

[38] Michael Tschannen, Olivier Bachem, and Mario Lucic. "Recent advances in autoencoder-based representation learning". In: *arXiv preprint arXiv:1812.05069* (2018).

[39] Zheng Yang et al. "Autoencoder-based representation learning and its application in intelligent fault diagnosis: A review". In: *Measurement* 189 (2022), p. 110460.

[40] Raghavendra Chalapathy and Sanjay Chawla. "Deep learning for anomaly detection: A survey". In: *arXiv preprint arXiv:1901.03407* (2019).

[41] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. "Sequence to sequence learning with neural networks". In: *Advances in neural information processing systems* 27 (2014).

[42] Danilo Rezende and Shakir Mohamed. "Variational inference with normalizing flows". In: *International conference on machine learning*. PMLR. 2015, pp. 1530–1538.

[43] Alec Radford, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks". In: *arXiv preprint arXiv:1511.06434* (2015).

[44] Diederik P Kingma and Max Welling. "Auto-encoding variational bayes". In: *arXiv preprint arXiv:1312.6114* (2013).

[45] Carl Doersch. "Tutorial on variational autoencoders". In: *arXiv preprint arXiv:1606.05908* (2016).

[46] Xue Ying. "An overview of overfitting and its solutions". In: *Journal of physics: Conference series*. Vol. 1168. IOP Publishing. 2019, p. 022022.

[47] Mikhail Belkin et al. "Reconciling modern machine-learning practice and the classical bias–variance trade-off". In: *Proceedings of the National Academy of Sciences* 116.32 (2019), pp. 15849–15854.

[48] Lutz Prechelt. "Early stopping—but when?" In: *Neural networks: tricks of the trade: second edition* (2012), pp. 53–67.

[49] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[50] Hui Zou and Trevor Hastie. "Regularization and variable selection via the elastic net". In: *Journal of the royal statistical society: series B (statistical methodology)* 67.2 (2005), pp. 301–320.

[51] Pascal Vincent et al. "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion." In: *Journal of machine learning research* 11.12 (2010).

[52] Pascal Vincent et al. "Extracting and composing robust features with denoising autoencoders". In: *Proceedings of the 25th international conference on Machine learning*. 2008, pp. 1096–1103.

[53] Xavier Bouthillier et al. "Dropout as data augmentation". In: *arXiv preprint arXiv:1506.08700* (2015).

[54] Jianglin Liang and Ruifang Liu. "Stacked denoising autoencoder and dropout together to prevent overfitting in deep neural network". In: *2015 8th international congress on image and signal processing (CISP)*. IEEE. 2015, pp. 697–701.

[55] Pierre Baldi and Peter J Sadowski. "Understanding dropout". In: *Advances in neural information processing systems* 26 (2013).

[56] Ling Huang et al. "In-network PCA and anomaly detection". In: *Advances in neural information processing systems* 19 (2006).

[57] Sasan Karamizadeh et al. "An overview of principal component analysis". In: *Journal of Signal and Information Processing* 4.3B (2013), p. 173.

[58] Jerome Fan, Suneel Upadhye, and Andrew Worster. "Understanding receiver operating characteristic (ROC) curves". In: *Canadian Journal of Emergency Medicine* 8.1 (2006), pp. 19–20.

[59] Dalwinder Singh and Birmohan Singh. "Investigating the impact of data normalization on classification performance". In: *Applied Soft Computing* 97 (2020), p. 105524.

[60] Peter F Odgaard and Kathryn E Johnson. "Wind turbine fault detection and fault tolerant control-an enhanced benchmark challenge". In: *2013 American Control Conference*. IEEE. 2013, pp. 4447–4452.

[61] Guoqian Jiang et al. "Wind turbine fault detection using a denoising autoencoder with temporal information". In: *IEEE/Asme transactions on mechatronics* 23.1 (2017), pp. 89–100.