AcousNomaly: Learning to Detect Anomalies in Acoustic Telemetry Data using Unsupervised Learning

A dissertation submitted in fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in the

DEPARTMENT OF MATHEMATICS
RHODES UNIVERSITY

by

Siphendulwe Zaza

Supervisor: Prof M. Atemkeng

Co-supervisor: Dr. T.S. Murray

January 2025

Declaration

I, Siphendulwe Zaza, hereby submit this thesis entitled *AcousNomaly: Learning to Detect Anomalies in Acoustic Telemetry Data using Unsupervised Learning*, in accordance with Rhodes University's plagiarism policy. I affirm that all content within this document is the result of my own original work. This thesis is submitted in fulfillment of the requirements for the degree of Master of Science in the Department of Mathematics at Rhodes University. It has not been submitted in its entirety or in part for any degree or diploma at this or any other university.

This research was conducted under the supervision of Prof M. Atemkeng and Dr. T.S. Murray. I sincerely endorse the academic standards and ethics of Rhodes University. Where applicable, any use of external ideas, concepts, data, or results has been properly acknowledged and cited.

I further declare that the reproduction and publication of this thesis by Rhodes University will not infringe on any third-party rights.

Signature:

Siphendulwe Zaza
Date: 19th April 2025

Abstract

Acoustic telemetry data plays a vital role in understanding the behaviour and movement of aquatic animals. However, these datasets, which often consist of millions of individual data points, frequently contain anomalous movements that pose significant challenges. Traditionally, anomalous movements are identified either manually or through basic statistical methods, approaches that are time-consuming and prone to high rates of unidentified anomalies in large datasets. This study focuses on the development of automated classifiers for a large telemetry dataset comprising detections from fifty acoustically tagged dusky kob (*Argyrosomus japonicus*) monitored in the Breede Estuary, South Africa. Using an array of 16 acoustic receivers deployed throughout the estuary between 2016 and 2021, resulting in the collection of over three million individual data points. This project presents detailed guidelines for data pre-processing, resampling strategies, labelling process, feature engineering, data splitting methodologies, and the selection and interpretation of unsupervised Machine Learning (ML) and Deep Learning (DL) models for anomaly detection. Among the evaluated models, Neural Network Autoencoder (NN-AE) demonstrated superior performance, aided by the threshold-finding algorithm proposed in this study. NN-AE achieved a high recall with no false normal (i.e., no misclassifications of anomalous movements as normal patterns), a critical factor in ensuring that no true anomalies are overlooked. In contrast, other models exhibited false normal fractions exceeding $\sim$ 0.9, indicating they failed to detect the majority of true anomalies—a significant limitation for telemetry studies where undetected anomalies can distort interpretations of movement patterns. While the NN-AE's performance highlights its reliability and robustness in detecting anomalies, it faced challenges in accurately learning normal movement patterns when these patterns gradually deviated from anomalous ones. To the best of our knowledge, this study represents the first effort to develop automated methods leveraging ML and DL to address anomalous detections in acoustic telemetry datasets.

Keywords: Acoustic telemetry, anomaly, *Argyrosomus japonicus*, machine learning, deep learning, Breede Estuary, dusky kob

Contents

List of Tables

List of Figures

List of Abbreviations

AEs: Autoencoders

AI: Artificial Intelligence

ANN: Artificial Neural Network

ATAP: Acoustic Tracking Array Platform

AUC: Area Under the Curve

BNN: Biological Neural Network

CNNs: Convolutional neural networks

DBSCAN: Density-based spatial clustering of applications with noise

DL: Deep Learning

DT: Decision Trees

FA: False Anomalies

FN: False Normals

IF: Isolation Forest

LOF: Local Outlier Factor

LSTM: Long Short-Term Memory

MAE: Mean Absolute Error

ME: Mean Error

ML: Machine Learning

MSE: Mean Squared Error

NN-AE: Neural Network Autoencoder

NN: Neural Network

ReLU: Rectified Linear Unit

RL: Reinforcement Learning

RNN: Recurrent Neural Network

ROC: Receiver Operating Characteristic

SSE: Sum of Squares Error

SVM: Support Vector Machine

TA: True Anomalies

TL: Total Length

TN: True Normals

Acknowledgements

I would like to extend my deepest gratitude to my supervisors, Prof M. Atemkeng and Dr. T.S. Murray. Their unwavering support, guidance, and encouragement have been instrumental in the completion of this thesis. Their attention to detail, patience, and profound expertise have not only enhanced the quality of this work but also significantly contributed to my personal and academic growth. Their ability to identify and address errors, inconsistencies, and challenges with care and precision has been invaluable throughout this journey. I am deeply grateful for their wisdom, understanding, and belief in my abilities, which have been a constant source of motivation.

I am deeply grateful to my late father, Mengezi Garnet Zaza, whose memory continues to inspire and guide me, and to my mother, Pumza Eunoria Zaza, for her unwavering support, immense sacrifices and unconditional love. Her selflessness and dedication to providing me with the best opportunities in life have been the foundation for my achievements. I would also like to thank my brothers and sister for their emotional support, encouragement and unwavering faith in me throughout this journey.

Special thanks to the Acoustic Tracking Array Platform, a marine infrastructure platform managed by the South African Institute for Aquatic Biodiversity (SAIAB) for providing the essential data that made this research possible, and to LEVENSTEIN, THE ADA & BERTIE BURSARY for the financial support that enabled my MSc studies.

I would like to acknowledge the members of the Rhodes Artificial Intelligence Research Group (RAIRG), including Nicole Oyetunji, Masixole Jojo, Siphelele Futhusi, Casey Chuma, Nkosinathi Ntuli, and others, for their insightful advice, encouragement, and constructive feedback, which significantly improved the quality and clarity of this work. My heartfelt thanks go to Sisipho Hamlomo for his time and effort in proofreading parts of this thesis and for his unwavering support in my academic journey and personal development. His encouragement and confidence in my abilities have been a constant source of motivation and inspiration over the years.

Lastly, I would like to extend my appreciation to everyone who has contributed to this journey in any way, big or small. Your support has made this achievement possible.

Publications derived from this thesis

1. Manuscript under review in *Ecological Informatics*

    - Zaza, S., Atemkeng, M., Murray, T.S., Filmalter J.D., & Cowley, P.D. *"Unsupervised anomaly detection in large-scale estuarine acoustic telemetry data"*.

Chapter 1

Introduction

1.1 Background

Studying fish movement and behaviour is crucial for understanding ecological dynamics and implementing effective conservation strategies (Crossin et al., 2017; Lowerre-Barbieri et al., 2019). Movements can be studied using various methods, including conventional dart tags providing low-resolution, coarse-scale data (Hughes et al., 2022), and acoustic telemetry which provides high-resolution, fine-scale data (Thorstad et al., 2013; Hussey et al., 2015). Acoustic telemetry, which involves recording signals from acoustic transmitters using deployed receivers, is currently the most popular method for studying the movements of teleosts and elasmobranchs (Hussey et al., 2015; Matley et al., 2022). While primarily applied to these groups, its use in marine mammals remains limited due to logistical and ethical challenges. This method provides valuable insights into movement patterns, habitat use, and survival of tagged individuals. Acoustic telemetry data, in essence, collect presence/absence information of tagged individuals. These data can be reduced to a time series, listing the dates and times tagged individuals (which have been tagged with acoustic transmitters that have unique ID codes) and the locations at which the detections occurred.

One of the biggest strengths of acoustic telemetry is the amount of data collected, which in some instances can reach millions of individual data points of numerous individuals from various species (Dhellemmes et al., 2023). However, this is also one of its biggest challenges. Large datasets have their own difficulties when it comes to analyses (Simpfendorfer et al., 2015). Associated with (but not limited to) large datasets are anomalous or false detections (Simpfendorfer et al., 2015; Dhellemmes et al., 2023). These anomalies can result from a variety of sources, including environmental interference such as the salinity or temperature of the water affecting signal transmission, or overlapping signals when multiple acoustic transmitters are in close proximity. Additionally, the collision of acoustic transmissions can create a detection of a different transmitter ID by the acoustic receiver, leading to

misidentified detections (Simpfendorfer et al., 2015). Another potential source of false detections is tag malfunction, where the tag is constantly sending erroneous signals. This work focuses on three specific anomalies defined in Section 3.6: (1) *An individual fish was recorded at only one station throughout the study period* (suggesting malfunction or predation), (2) *An individual fish moved as per normal, but then remained at the same station for more than 120 days*, and (3) *An individual fish was not detected by consecutive stations and the individual was missed by more than one consecutive station*. These anomalies can significantly skew the interpretation of fish movements and behaviour patterns (Chambert et al., 2015; Simpfendorfer et al., 2015), making it essential to develop a strong anomaly detection system to ensure data integrity. In smaller telemetry datasets, anomalies can often be visually identified after plotting the data. However, this method is far too time-consuming for larger datasets comprising millions of detections.

Several statistical packages have been developed that filter, process and analyse passive acoustic telemetry data in the statistical environment R (R Core Team, 2022), including *rsp* (Niella et al., 2020), *actel* (Flávio et al., 2021), *ATfiltR* (Dhellemmes et al., 2023), and *telemetR* (Spaulding, 2024). While each have their strengths, another option to process large quantities of detection data is through Machine Learning (ML). ML refers to a subfield of Artificial Intelligence (AI) that focuses on the development of algorithms that can learn patterns from data and make decisions or predictions without being explicitly programmed (Helm et al., 2020). Rather than relying on hard-coded rules, ML systems improve their performance over time by exposing themselves to new data, allowing them to adapt to complex, dynamic environments. DL, a specialized subfield within ML, uses artificial neural networks with multiple layers often referred to as deep neural networks to model complex and abstract representations of data (Sarker, 2021). These deep architectures enable DL models to automatically extract hierarchical features from raw inputs, making them particularly effective for high-dimensional and unstructured data such as images, audio and text. While traditional (or shallow) neural networks consist of one or two layers and can model relatively simple relationships, deep neural networks consist of many layers, allowing them to model more abstract and complex patterns.

In ecological research, these methods have been successfully applied across various environments. For example, automating the detection of Hainan gibbon *Nomascus hainanus* calls using deep Neural Network (NN) (Dufourq et al., 2021), and monitoring population numbers of wildebeest and zebra using a U-Net-based DL model integrated with a post-processing clustering model (Wu et al., 2023). In the aquatic environment, ML and DL models have been applied to underwater video analysis to detect atypical fish behaviour (Wang et al., 2020), and while their potential in acoustic telemetry remains relatively under explored, these models are slowly being incorporated into telemetry studies. For example, the migration fate of Atlantic salmon *Salmo salar* smolts was classified using unsupervised k-means cluster analyses and supervised random forest models (Notte et al., 2022), predatory

fishes have been identified through the use of predation tags and associated movements using random forest algorithms (Klinard et al., 2020), relative habitat selection by multiple shark species was predicted using random forest models (Griffin et al., 2021), and habitat suitability of two estuarine species was modeled using deep feed-forward Artificial Neural Network (ANN) (Guénard et al., 2020).

In ML, there are two main approaches for anomaly detection: supervised and unsupervised models. These methods will be discussed in detail in Chapter 2. Supervised ML methods are effective in many areas, they are less suitable for this study due to the highly imbalanced nature of the telemetry datasets, where a significantly higher proportion of normal detections are present compared to anomalies, making it difficult to train supervised models without overfitting to the majority class. This limitation restricts the practicality of supervised models for this study. Unsupervised anomaly detection techniques, such as Isolation Forest (IF), Density-based spatial clustering of applications with noise (DBSCAN), and Local Outlier Factor (LOF), have been widely used to detect anomalies in time series data (Schindler et al., 2023). While these methods are effective, they often struggle to capture complex temporal dependencies inherent in telemetry data (Benova & Hudec, 2024). However, DL approaches such as the NN Autoencoder (NN-AE) excel at modeling such dependencies, allowing the detection of subtle deviations in large and intricate datasets (Chen et al., 2021). In the context of big data, where traditional statistical and supervised methods may fall short, unsupervised classifiers offer a robust alternative to ensure data integrity by significantly improving the accuracy of detecting false detections (Chen et al., 2021). This study addresses the gap by detailing the training of unsupervised classifiers for the accurate detection of false movements of an important estuary-dependent species, dusky kob (*Argyrosomus japonicus*), within a permanently open estuary along South Africa's southern coast. The purpose of this project is to provide telemetry ecologists with a comprehensive understanding of the training process, the decision-making involved in model selection, and practical guidelines for these decisions.

1.2 Problem Statement

Analysing large-scale acoustic telemetry data presents a challenge in detecting and managing anomalies. The data often contains erroneous detections caused by environmental noise, tag malfunctions or overlapping signals. The data often contain two types of anomalies: technical errors (e.g., environmental noise, tag malfunctions, or colliding signals), which are removed during pre-processing; and behavioural anomalies (e.g., unusual movement patterns defined in Section 3.6), which are the focus of this study. This work specifically investigates how well unsupervised ML models can detect behavioural anomalies in movement data, rather than raw detection errors such as false IDs, tag malfunctions, or colliding signals. If left undetected, these anomalies can distort ecological interpretations of fish behaviour and

habitat use. Traditionally, researchers identify anomalies manually by visually inspecting the data. While this is feasible for small datasets, it becomes impractical for datasets with millions of detections due to time constraints and inefficiencies. In addition, statistical anomaly detection methods struggle to capture complex movement patterns over time, as temporal relationships in the data are often not taken into account. Therefore, an automated method for detecting anomalies in large telemetry datasets is urgently needed to ensure the quality and reliability of the data. This study addresses these challenges by proposing an unsupervised framework using IF, DBSCAN, LOF, and NN-AE to detect anomalies in the telemetry data of dusky kob (*Argyrosomus japonicus*) from Breede Estuary, South Africa.

1.3 Objectives

The aim of this work is to detect anomalies in acoustic telemetry data from the Breede Estuary using ML and DL techniques. The approach involves gathering and pre-processing of acoustic telemetry data, including handling missing values, scaling and performing feature engineering to prepare it for analysis. Unsupervised DL models, such as NN-AE, are developed to identify anomalies by optimising thresholds based on reconstruction errors. While NN-AE falls under DL due to its multilayered neural network architecture, it is applied here within the broader context of unsupervised ML. The performance of NN-AE is compared to traditional anomaly detection models such as IF, DBSCAN and LOF using metrics such as accuracy, precision, recall, f_1-score, specificity and AUC. Temporal dependencies in fish movement data are taken into account by using DL models to capture sequential patterns. Identified anomalies are analysed to gain ecological insights into the movements and behaviour of dusky kob, contributing to its conservation. The framework is designed to be scalable, enabling efficient processing of large telemetry datasets while minimising manual intervention. Clear documentation and guidelines are provided to assist telemetry ecologists in pre-processing data, selecting models, and effectively evaluating results.

1.4 Contributions

Although much research has been conducted in the field of ML and AI on anomaly detection, relatively little attention has been paid to its application to aquatic telemetry data. This study addresses this gap by developing an unsupervised anomaly detection system based on a NN-AE architecture. The system is designed to learn normal movement patterns from telemetry datasets and accurately distinguish them from unusual behaviours. By automating this process, the system improves the efficiency and reliability of analysing telemetry data on a large scale. Unlike traditional statistical methods, which can be labour intensive and error-prone, this approach minimises manual effort while ensuring high data integrity. Ultimately, the framework improves the quality of ecological insights and promotes data-driven conservation efforts through accurate and scalable anomaly detection.

1.5 Thesis Outline

Chapter 1 discusses acoustic telemetry, the challenges of anomaly detection in large datasets, and the objectives of automating the process using ML techniques like NN-AE.

Chapter 2 discusses the fundamentals of ML and DL, including various algorithms like perceptrons, CNNs, LSTMs, and AEs, and their applications in anomaly detection.

Chapter 3 discusses the acoustic telemetry dataset from the Breede Estuary, detailing the data collection, data pre-processing, standardization, and spatial visualization techniques used in the study.

Chapter 4 discusses the methodology for anomaly detection, focusing on the implementation of NN-AE and other models like IF, DBSCAN, and LOF, along with model evaluation metrics.

Chapter 5 discusses the results of the study, comparing the performance of different models. It also discusses the key findings, limitations of the current models, and provides suggestions for future work.

Chapter 6 summarises the work, highlights its limitations, and outlines future directions.

Chapter 2

Machine Learning and Deep Learning

Sections 2.1 to 2.8 of this chapter introduce different ML and DL algorithms relevant for anomaly detection tasks. These include foundational concepts such as Biological Neural Network (BNN), and ANN, alongside more advanced architectures like Convolutional neural networks (CNNs) and Long Short-Term Memory (LSTM) networks. Section 2.9 provides a detailed overview of unsupervised learning methods, which are particularly suitable for anomaly detection in large, unlabelled datasets. Techniques such as IF, LOF, DBSCAN, and Autoencoders (AEs) are explored, specifically emphasizing their ability to detect anomalies in complex telemetry data. Finally, Section 2.10 discusses Reinforcement Learning (RL) concepts and their potential role in optimizing decision-making processes for anomaly detection systems.

2.1 Introduction

As discussed in Chapter 1, ML is a branch of AI that enables systems to automatically learn from data and improve their performance on specific tasks without being explicitly programmed (Mitchell, 1997; Murphy, 2012). Instead of relying on rule-based approaches, ML algorithms identify patterns in the data. This makes them particularly suitable for analyzing complex and high-dimensional ecological datasets, such as those generated by acoustic telemetry. Depending on the type of data available, ML algorithms can be trained on labelled data (supervised learning), unlabelled data (unsupervised learning), a combination of both (semi-supervised learning), or through trial-and-error interactions (reinforcement learning). ML is used in many areas, such as finance (Hamlomo, 2021), healthcare (Habehh & Gohel, 2021), marketing (Ngai & Wu, 2022) and autonomous systems (Shah, 2020), and is crucial for tasks such as image and speech recognition, natural language processing, recommendation systems, and anomaly detection (Jordan & Mitchell, 2015). By taking advantage of ML, companies can gain new insights, drive innovation, and automate complex tasks, leading to more efficient decision-making. A practical example of the automation of complex tasks

using AI can be found in the healthcare sector, where AI-supported systems help diagnose diseases (Maleki Varnosfaderani & Forouzanfar, 2024). For example, AI algorithms analyse medical images such as X-rays or MRIs to detect anomalies such as tumors or bone fractures (Nhlapho et al., 2024; Brima & Atemkeng, 2024). These systems, which have been trained on huge datasets of labelled images, can often detect problems with an accuracy comparable to that of experienced radiologists, saving time and helping doctors make faster and more accurate diagnoses (Alowais et al., 2023). This not only increases efficiency but also improves patient outcomes by enabling early intervention.

2.2 Biological Neural Networks

A BNN is a collection of interconnected neurons, as shown in Figure 2.1. This network is studied to gain insights into the structure and function of nervous systems (Sterratt et al., 2023). It consists of approximately more than 86 billion neurons in the human brain, each of which processes and transmits information. The neurons communicate through electrical impulses sent through dendrites, synapses, and axons. This complex network enables calculations, perceptions, and functions of the nervous system. Each neuron can be connected to many others, forming an extensive network. The connections, or synapses, are typically formed from the axons to the dendrites and facilitate communication. BNNs inspire AI, cognitive modeling, and ANN.

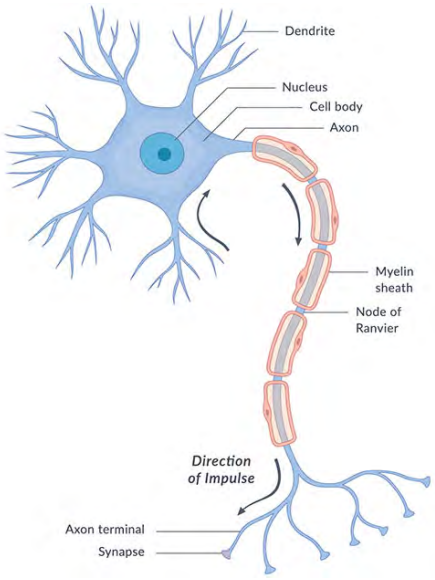


Figure 2.1: This figure shows the structure of a biological neuron, showcasing its key components: dendrites, nucleus, cell body, axon, myelin sheath, nodes of Ranvier, axon terminals, and synapse (NICHD, 2018).

2.2.1 Perceptrons

A perceptron is a fundamental mathematical model that simulates the behaviour of a biological neuron (Shanmuganathan, 2016). Biological neurons receive signals, process them,

and decide whether to pass them on to other neurons. Similarly, a perceptron receives input data, processes it through weights and biases, and makes a decision using an activation function. The perceptron computes the weighted sum of its inputs and a bias term, that is:

$$Z = \sum_{i=1}^{n} w_i x_i + b, \tag{2.2.1}$$

where x_i are the input features, w_i are the corresponding weights, b is the bias term, and n is the number of inputs. This weighted sum is then passed through an activation function σ, which determines the perceptron's output:

$$\hat{y} = \sigma(Z). \tag{2.2.2}$$

Figure 2.2 shows the typical architecture of a perceptron.

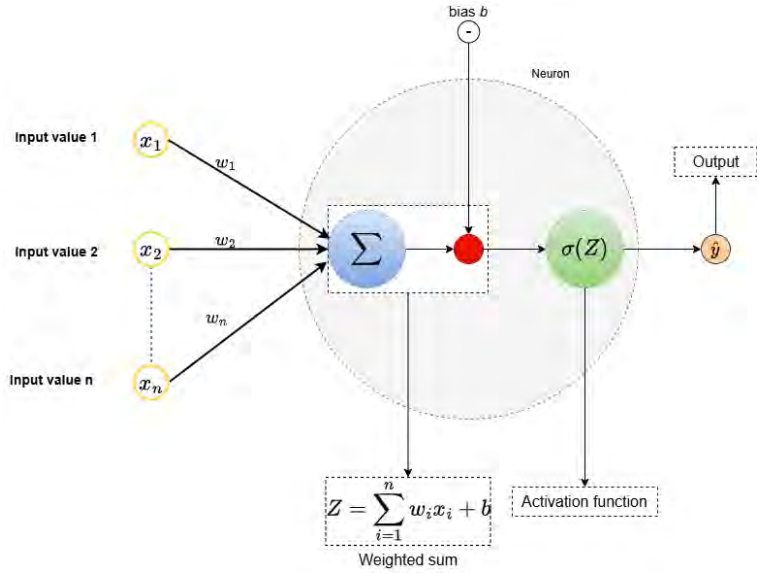


Figure 2.2: An illustration of an artificial neuron (perceptron). Inputs are weighted, summed with a bias, and passed through an activation function to produce an output, modeling the decision-making mechanism of NNs (adapted from Staudemeyer & Morris (2019)).

2.3 Artificial Neural Networks

ANNs are a class of ML models inspired by the biological neuron and the function of the human brain (Nwadiugwu, 2020). While perceptrons mimic basic neuron behaviour by processing inputs through weights, summation, and activation, ANNs extend this concept through layered architectures. ANNs consist of interconnected neurons organized in layers: an input layer, one or more hidden layers, and an output layer. Unlike simple perceptrons, ANNs can learn complex patterns and relationships due to their multiple layers and non-linear activation functions.

2.3.1 A Simple Feed-Forward Neural Network

To extend beyond the perceptron, this project presents a simple Feed-Forward NN architecture (FNN). The computation in a single neuron in the hidden layer of the FNN can be expressed as:

$$Z_k = \sum_{i=1}^{n} w_{ik} x_i + b_k, \tag{2.3.1}$$

where w_{ik} represents the weight connecting the i-th input to the k-th neuron in the hidden layer, and b_k is the bias term for the k-th neuron. The activation function σ is applied to Z_k to compute the output of the neuron:

$$h_k = \sigma(Z_k), \tag{2.3.2}$$

where h_k represents the activation output of the k-th neuron in the hidden layer and becomes an input to the next layer in the network.

For the output layer, the computation for a single output neuron is:

$$\hat{y}_j = \sigma\left(\sum_{k=1}^{m} w_{kj} h_k + b_j\right), \tag{2.3.3}$$

where w_{kj} represents the weight connecting the k-th hidden neuron to the j-th output neuron, and b_j is the bias term for the j-th output neuron. Figure 2.3 shows an example of a basic feed-forward neural network.

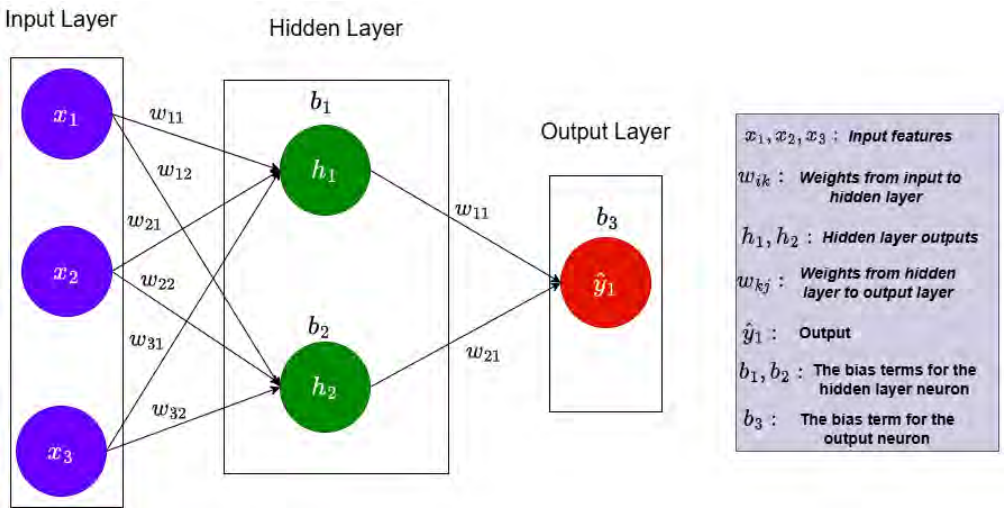


Figure 2.3: A basic feed-forward neural network consists of three input features (x_1, x_2, x_3), one hidden layer with two nodes (h_1, h_2), and a single output (y_1). The weights (w_{ik}) connect the input layer to the hidden layer, and the weights (w_{kj}) connect the hidden layer to the output layer, along with bias terms. The bias terms for the hidden layer neurons are (b_1, b_2), and the bias term for the output layer neuron is (b_3). Each node in the hidden and output layers applies an activation function to transform inputs and generate predictions (adapted from AIML.com research (2024)).

2.3.2 Structure of ANNs

ANNs are composed of three primary layers: the input layer, hidden layers, and the output layer. These layers work together to process data and generate predictions. The input layer is responsible for mapping the input features $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ directly to the neurons. Each neuron in the input layer corresponds to a feature in the data, and the input is passed unchanged to the subsequent layer. In the hidden layers, the neurons compute the weighted sum of their inputs, add a bias term, and apply an activation function to introduce non-linearity. This process allows the network to model complex relationships in the data. The calculation for the hidden neurons can be expressed as follows:

$$h_j = \sigma\left(\sum_{i=1}^{n} w_{ij} x_i + b_j\right), \tag{2.3.4}$$

where w_{ij} is the weight connecting the i-th input to the j-th hidden neuron, b_j is the bias term for the j-th hidden neuron, and σ is the activation function. This formulation is consistent with the general computation described earlier in Equation 2.3.2. The output layer generates predictions based on the transformed features from the hidden layers. For classification tasks, the softmax activation function is often used to compute class probabilities:

$$\hat{y}_k = \text{softmax}\left(\sum_{j=1}^{m} w_{jk} h_j + b_k\right), \tag{2.3.5}$$

where w_{jk} is the weight connecting the j-th hidden neuron to the k-th output neuron, b_k is the bias term for the k-th output neuron, and $\hat{y}_k$ is the predicted probability for class k. This layered structure enables ANNs to approximate complex, non-linear mappings between input features and output predictions, making them powerful tools for tasks like classification, regression, and anomaly detection.

2.4 Activation Functions

Activation functions introduce non-linearity into the NN, enabling it to learn and model complex relationships in the data. Without activation functions, a NN would act as a linear classifier regardless of the number of layers. This section provides an overview of the following activation functions: sigmoid, tanh, Relu, Leaky Relu, and the Softmax activation function.

2.4.1 Sigmoid

The sigmoid function maps input values to a range between 0 and 1, making it suitable for binary classification tasks. However, it suffers from vanishing gradients for extreme input values (Sharma et al., 2017), which can hinder learning in deep NNs. The sigmoid function

is defined mathematically as:

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{2.4.1}$$

The graphical representation of the sigmoid activation function is shown in Figure 2.4. The graph of the sigmoid function demonstrates how it maps any real-valued input to a range between 0 and 1. The curve is S-shaped, with the function asymptotically approaching 0 for very negative inputs and 1 for very positive inputs.

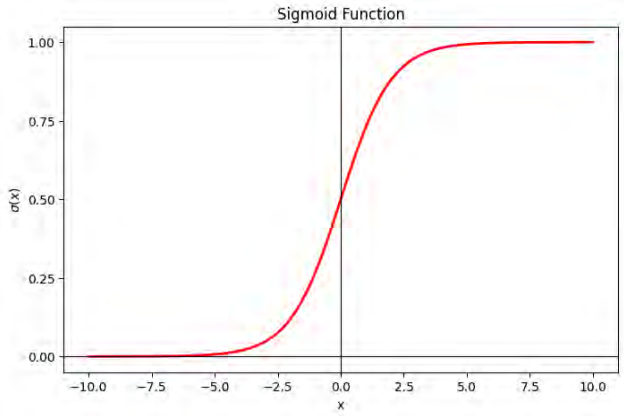


Figure 2.4: Sigmoid Function

2.4.2 Tanh

Tanh squashes input values to a range between -1 and 1, providing better gradients during training compared to the sigmoid function (Sharma et al., 2017). This helps in reducing the issue of vanishing gradients, particularly for deeper NNs. The Tanh function is defined mathematically as:

$$\sigma(x) = \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}. \tag{2.4.2}$$

The graphical representation of the Tanh activation function is shown in Figure 2.5. The Tanh function graph resembles the sigmoid function but maps inputs to a range between -1 and 1. The function is symmetrical about the origin, making it zero-centered.

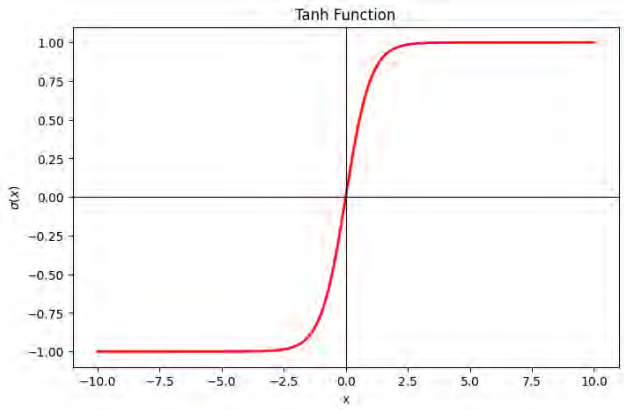


Figure 2.5: Tanh Function

2.4.3 Rectified Linear Unit (ReLU)

Rectified Linear Unit (ReLU) is a widely used activation function due to its simplicity and efficiency. It introduces non-linearity while allowing positive values to pass through unchanged, which accelerates training and helps mitigate the vanishing gradient problem (Ding et al., 2018). However, ReLU can suffer from the dying ReLU problem where neurons output zero for all inputs. The ReLu function is defined mathematically as:

$$\sigma(x) = \max(0, x). \tag{2.4.3}$$

The graphical representation of the ReLU activation function is shown in Figure 2.6. The ReLU function graph shows a linear relationship for positive inputs and outputs zero for negative inputs.

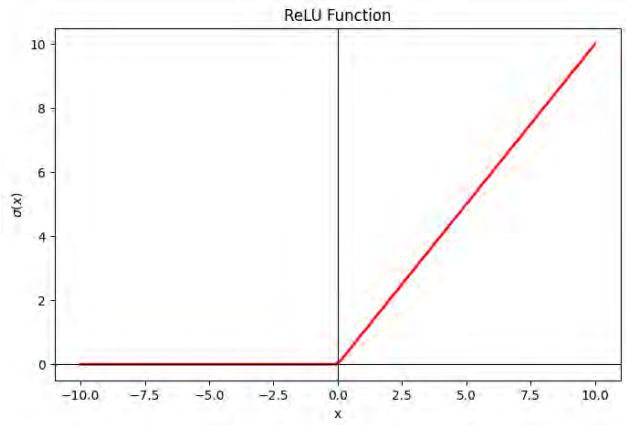


Figure 2.6: ReLU Function

2.4.4 Leaky ReLU

Leaky ReLU is a variation of the standard ReLU activation function designed to address the dying ReLU problem during training, where some units stop contributing to the learning process by outputting zero for all inputs. Unlike the standard ReLU, which completely zeros out negative values, Leaky ReLU introduces a small, non-zero gradient for negative inputs, ensuring that neurons continue to learn and update their weights even when the input is negative (Ding et al., 2018). The parameter α controls the slope for negative inputs. The Leaky ReLU function is defined mathematically as:

$$\sigma(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0. \end{cases} \tag{2.4.4}$$

The graphical representation of the Leaky ReLU activation function is shown in Figure 2.7. The graph of the Leaky ReLU function modifies the ReLU by introducing a small slope (α) for negative inputs. The figure highlights how Leaky ReLU addresses the dying ReLU

problem by allowing a non-zero gradient for negative inputs.

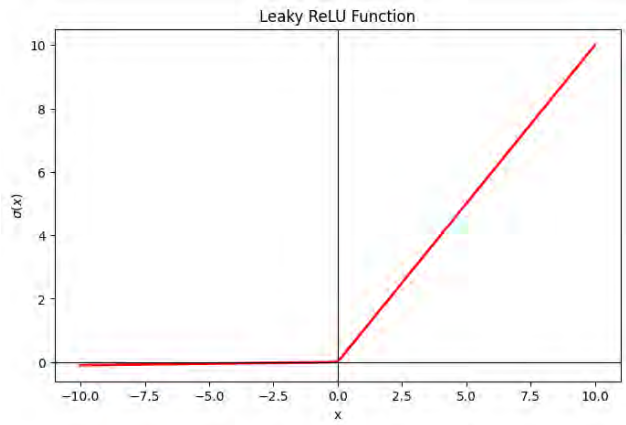


Figure 2.7: Leaky ReLU Function

2.4.5 Softmax

The Softmax function is commonly applied in the output layer of classification networks, especially for multi-class classification tasks, where the goal is to assign data points to one of several categories (Sharma et al., 2017). It transforms raw scores, also known as logits, into a probability distribution where the probabilities for all classes sum to 1. The Softmax function is defined mathematically as:

$$\sigma(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}. \tag{2.4.5}$$

The graphical representation of the Softmax activation function is shown in Figure 2.8. The Softmax function graph shows how input logits are transformed into probabilities that sum to 1. This transformation is achieved through exponential scaling, which amplifies the highest input values, causing them to dominate the output probabilities.

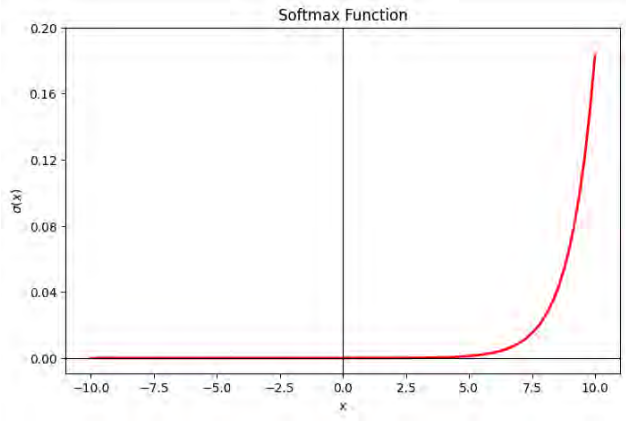


Figure 2.8: Softmax Function

2.5 Supervised Machine Learning

In supervised ML, an algorithm is trained with labelled data, where each data point contains both the input features and the corresponding class label. The model learns to match the inputs to the outputs so that it can make predictions for new data (Nasteski, 2017). Commonly used supervised learning algorithms include linear regression, logistic regression, Support Vector Machine (SVM), Decision Trees (DT) and ANN. These models are trained by minimising the loss function, which measures the difference between the predicted and actual outputs.

A major challenge in supervised learning is overfitting, where a model learns the noise and details in the training data so well that it performs exceptionally well on the training dataset, but cannot generalise to unseen data (Ying, 2019). This usually happens when the model becomes too complex and captures irrelevant patterns or noise instead of the underlying trends. To address overfitting, techniques such as cross-validation, regularisation, pruning, or early stopping are applied to improve the model's ability to generalise to new data (Kernbach & Staartjes, 2022).

In this work, supervised learning methods were not used due to the nature of the acoustic telemetry dataset, where the vast majority of data points are unlabelled. Supervised models, such as DT require labelled datasets for training which is not feasible in this context, as manually labelling anomalous detections in a dataset with millions of data points would be extremely time and resource consuming. Moreover, the imbalance between normal data points and anomalies makes supervised learning less effective without sophisticated balancing techniques. In contrast, unsupervised models are better suited for this task because they can detect hidden patterns and anomalies in the data without requiring labelled inputs. This makes them ideal for applications such as acoustic telemetry, where anomalies occur rarely and are often unknown ahead of time.

Supervised learning is mathematically described by assuming the input space $X \subseteq \mathbb{R}^n$, where each observation $\mathbf{x}_i = (x_1, x_2, \ldots, x_n) \in X$ is associated with a corresponding class label $y_i \in Y$, where $Y \subseteq \{c_1, c_2, \ldots, c_M\}$ for classification tasks (discrete classes) or $Y \subseteq \mathbb{R}$ for regression (continuous output). Given a dataset $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$, supervised learning seeks to find a function $f \in \mathcal{F}$ such that:

$$f : X \to Y, \tag{2.5.1}$$

where $\mathcal{F}$ represents the family of candidate functions from which the supervised learning algorithm selects the optimal function f. The objective of supervised learning is to minimise the expected risk:

$$R(f) = \mathbb{E}_P \left[L(y, f(\mathbf{x})) \right], \tag{2.5.2}$$

where $L(\cdot)$ is a loss function, such as Mean Squared Error (MSE) for regression problems or cross-entropy loss for classification problems. Since the true distribution $\mathcal{P}$ is unknown, the

empirical risk is used as an approximation:

$$\begin{aligned} R_D(f) &= \mathbb{E}_D\left[L(y, f(\mathbf{x}))\right] \\ &= \frac{1}{N}\sum_{i=1}^{N} L(y_i, f(\mathbf{x}_i)). \end{aligned} \tag{2.5.3}$$

The objective is to minimise this empirical risk using optimization methods such as gradient descent, assuming that $R_D(f)$ is a good approximation of $R(f)$; i.e. $R_D(f) \approx R(f)$.

2.5.1 Loss for Classification Problems

Loss functions are a fundamental component of ML models. They guide the optimization process by quantifying the difference between the model's predictions and the true labels. For classification tasks, the choice of loss function depends on the nature of the problem, in particular the type and number of classes. The most commonly used loss functions for classification problems are binary cross-entropy and categorical cross-entropy (Ruby & Yendapalli, 2020).

Binary cross-entropy loss is used for binary classification tasks where the class label $y_i \in \{0, 1\}$. The predicted probability $f(\mathbf{x}_i)$ is typically obtained from a sigmoid activation function and represents the likelihood of the positive class. The binary cross-entropy loss function is given as:

$$L(\mathbf{x}_i, y_i, f) = -\left[y_i \log f(\mathbf{x}_i) + (1 - y_i)\log(1 - f(\mathbf{x}_i))\right]. \tag{2.5.4}$$

On the other hand, the categorical cross-entropy loss is used for classification problems where $Y = \{c_1, c_2, \ldots, c_C\}$. In this case, the class label (y_i) is typically represented as a one-hot encoded vector, and $f(\mathbf{x}_i)$ is the model's predicted probability distribution over the C classes. The categorical cross-entropy loss measures the deviation between the predicted probabilities $f(\mathbf{x}_i)$ and the actual class label y_i. It is defined as:

$$L(\mathbf{x}_i, y_i, f) = -\sum_{j=1}^{C} y_{ij} \log f_j(\mathbf{x}_i), \tag{2.5.5}$$

where y_{ij} is 1 if the true label of $\mathbf{x}_i$ corresponds to class j, and 0 otherwise, and $f_j(\mathbf{x}_i)$ is the predicted probability for class j, computed by the model f.

2.5.2 Loss for Regression Problems

Loss functions for regression tasks measure the error between the predicted values and the actual target values. The most commonly used loss functions for regression problems include Mean Error (ME), Mean Absolute Error (MAE), and the Sum of Squares Error (SSE).

ME is used in regression problems where $Y \subseteq \mathbb{R}$ to measure the average signed error between the predicted values $f(\mathbf{x}_i)$ and the true target values y_i. It is computed as:

$$L(\mathbf{x}_i, y_i, f) = f(\mathbf{x}_i) - y_i, \tag{2.5.6}$$

where $f(\mathbf{x}_i)$ represents the predicted value for the observation $\mathbf{x}_i$, computed by the model f, y_i is the actual target value, and the term $f(\mathbf{x}_i) - y_i$ quantifies the signed error for a single observation. On the other hand, the MSE also known as the quadratic loss function, measures the average squared difference between the predicted values $f(\mathbf{x}_i)$ and the true target values y_i. It is defined as:

$$L(\mathbf{x}_i, y_i, f) = (f(\mathbf{x}_i) - y_i)^2, \tag{2.5.7}$$

where $f(\mathbf{x}_i)$ is the predicted value for the observation $\mathbf{x}_i$, y_i is the actual target value, and $(f(\mathbf{x}_i) - y_i)^2$ quantifies the squared error for a single observation. MAE quantifies the average absolute difference between the predicted values $f(\mathbf{x}_i)$ and the true target values y_i. It is expressed as:

$$L(\mathbf{x}_i, y_i, f) = |f(\mathbf{x}_i) - y_i|, \tag{2.5.8}$$

where $|f(\mathbf{x}_i) - y_i|$ measures the absolute error for a single observation. Unlike MSE, the MAE is more robust to outliers due to the linear nature of the absolute error (Willmott & Matsuura, 2005). In contrast, the SSE measures the total squared difference between the predicted values $f(\mathbf{x}_i)$ and the true target values y_i across all observations. It is computed as:

$$L(\mathbf{x}_i, y_i, f) = \sum_{i=1}^{N} (f(\mathbf{x}_i) - y_i)^2, \tag{2.5.9}$$

where $f(\mathbf{x}_i)$ represents the predicted value for the observation $\mathbf{x}_i$, y_i is the true target value, and $\sum_{i=1}^{N} (f(\mathbf{x}_i) - y_i)^2$ aggregates the squared errors across all N observations.

Each loss function has its strengths and weaknesses. ME for example, is useful for assessing the bias of predictions, while MSE emphasizes larger errors. MAE, on the other hand, provides robustness to outliers, and SSE provides an aggregate error measure for all observations. The choice of loss function depends on the specific requirements and objectives of the regression task.

2.6 The Feed-forward Process

In a NN, the feed-forward process is the way inputs move through the layers to produce the final output. Starting with an input vector $\mathbf{x}$, the data passes through each layer in order. At each layer, the input is transformed using a combination of mathematical operations, like adding weights and biases, followed by applying an activation function. This process repeats for all L layers of the network, eventually producing the output, $\hat{\mathbf{x}}$. Figure 2.9 shows the feed-forward and back-propagation calculations.

At the first layer, the input $\mathbf{x}$ undergoes a linear transformation through a combination of weights and biases:

$$Z^{[1]} = W^{[1]}\mathbf{x} + b^{[1]}, \tag{2.6.1}$$

where $W^{[1]} \in R^{n_1 \times n_0}$ is the weight matrix for the first layer, and $b^{[1]} \in R^{n_1}$ is the bias vector. Here, n_0 and n_1 are the number of neurons in the input layer and the first hidden layer, respectively. The result $Z^{[1]}$ represents the linear transformation of the input.

A non-linear activation function σ is then applied element-wise to $Z^{[1]}$, producing the activation output:

$$Y^{[1]} = \sigma(Z^{[1]}). \tag{2.6.2}$$

This activation introduces non-linearity, enabling the network to approximate complex functions. The activations $Y^{[1]}$ are then passed to the next layer, where the process is repeated:

$$Z^{[2]} = W^{[2]}Y^{[1]} + b^{[2]}, \tag{2.6.3}$$

$$Y^{[2]} = \sigma(Z^{[2]}). \tag{2.6.4}$$

Here, $W^{[2]} \in R^{n_2 \times n_1}$ and $b^{[2]} \in \mathbb{R}^{n_2}$ are the weights and biases of the second layer, and $Z^{[2]}$ is the pre-activation result.

For a network with L layers, this process generalizes to:

$$Z^{[l]} = W^{[l]}Y^{[l-1]} + b^{[l]}, \quad l = 1, 2, \ldots, L, \tag{2.6.5}$$

$$Y^{[l]} = \sigma(Z^{[l]}), \quad l = 1, 2, \ldots, L-1. \tag{2.6.6}$$

At the final layer, the output $Z^{[l]}$ may be processed differently, depending on the task. The feed-forward process concludes with a loss function as discussed in Eq. 2.5.3.

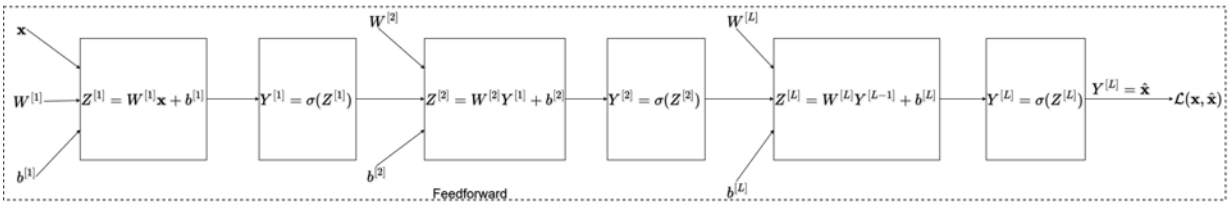


Figure 2.9: The feed-forward step propagates input $\mathbf{x}$ through layers of weights W and biases b, applying activation functions σ to produce the output $\hat{x}$.(adapted from Nandutu (2023))

2.7 The Back-propagation

Back-propagation is a fundamental algorithm for training NNs, designed to minimise errors by systematically adjusting the network's weights and biases. It calculates the gradient

of the loss function with respect to each weight using the chain rule, enabling efficient gradient-based optimisation (Hong, 2023). The process begins at the output layer, where the error between the predicted and actual values is measured. Gradients are then calculated and propagated backward through the network, updating each layer's parameters step by step. Figure 2.10 illustrates this process, highlighting how back-propagation uses the chain rule of differentiation to optimise the network's performance.

Using gradient descent, the partial derivative of the loss function with respect to the weights in the final layer $W_{kj}^{[L]}$ is:

$$\frac{\partial \mathcal{L}}{\partial W_{kj}^{[L]}} = \frac{\partial \mathcal{L}}{\partial Y_i^{[L]}} \frac{\partial Y_i^{[L]}}{\partial Z_i^{[L]}} \frac{\partial Z_i^{[L]}}{\partial W_{kj}^{[L]}}. \tag{2.7.1}$$

The chain rule allows these derivatives to be computed step-by-step. The partial derivative of the linear output $Z_i^{[L]}$ with respect to the weight $W_{kj}^{[L]}$ is:

$$\frac{\partial Z_i^{[L]}}{\partial W_{kj}^{[L]}} = Y_j^{[L-1]}, \tag{2.7.2}$$

where $Y_j^{[L-1]}$ is the activation from the previous layer.

The derivative of the activation function σ applied at the output layer is:

$$\frac{\partial Y_i^{[L]}}{\partial Z_i^{[L]}} = \sigma'(Z_i^{[L]}), \tag{2.7.3}$$

where $\sigma'(Z_i^{[L]})$ is the derivative of the activation function evaluated at $Z_i^{[L]}$.

The gradient of the loss function with respect to the output $Y_i^{[L]}$ is given by:

$$\frac{\partial \mathcal{L}}{\partial Y_i^{[L]}} = -(x_i - \hat{x}_i). \tag{2.7.4}$$

Substituting these terms, the gradient of the loss function with respect to $W_{kj}^{[L]}$ becomes:

$$\frac{\partial \mathcal{L}}{\partial W_{kj}^{[L]}} = -(x_i - \hat{x}_i)\sigma'(Z_i^{[L]})Y_j^{[L-1]}. \tag{2.7.5}$$

The same procedure is applied to compute gradients with respect to biases:

$$\frac{\partial \mathcal{L}}{\partial b_k^{[L]}} = -(x_i - \hat{x}_i)\sigma'(Z_i^{[L]}). \tag{2.7.6}$$

For earlier layers $l < L$, the error term propagates backward through the network. The

derivative of the loss function with respect to weights in layer l is computed as:

$$\frac{\partial \mathcal{L}}{\partial W_{kj}^{[l]}} = \delta_i^{[l]} Y_j^{[l-1]}, \tag{2.7.7}$$

where the error term $\delta_i^{[l]}$ is defined recursively as:

$$\delta_i^{[l]} = \left(\sum_k \delta_k^{[l+1]} w_{ik}^{[l+1]} \right) \sigma'(Z_i^{[l]}). \tag{2.7.8}$$

The weights and biases are updated iteratively using gradient descent:

$$W_t^{[l+1]} = W_t^{[l]} - \lambda \nabla_t W^{[l]}, \tag{2.7.9}$$

$$b_t^{[l+1]} = b_t^{[l]} - \lambda \nabla_t b^{[l]}, \tag{2.7.10}$$

where λ is the learning rate, and $\nabla_t W^{[l]}$ and $\nabla_t b^{[l]}$ represent the gradients at iteration t.

This iterative optimisation process minimises the reconstruction loss, allowing the network to learn meaningful patterns from the data and adjust its internal parameters for improved predictive accuracy.

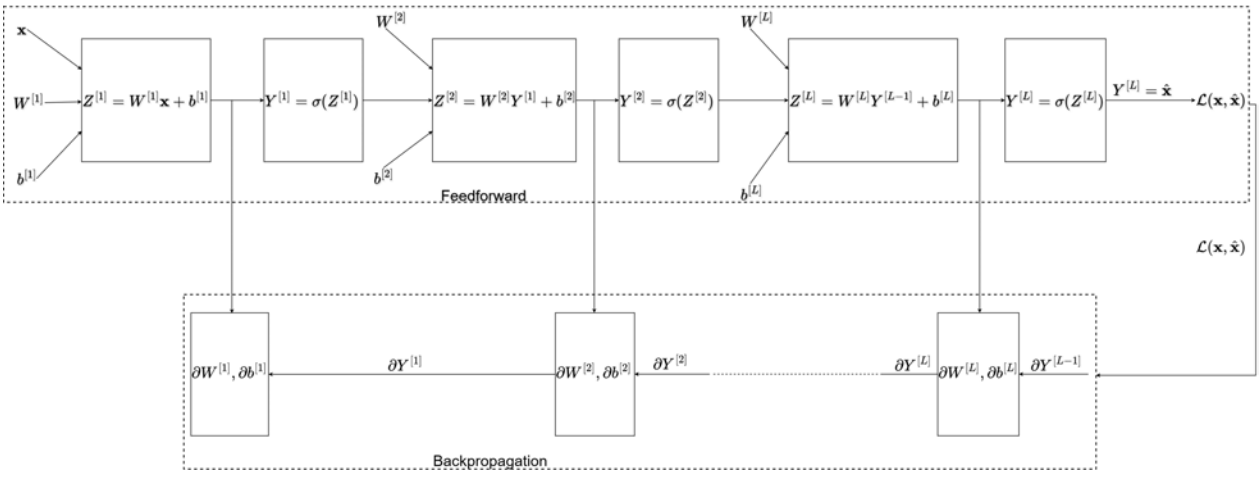


Figure 2.10: The feed-forward step propagates input $\mathbf{x}$ through layers of weights W and biases b, applying activation functions σ to produce the output $\hat{\mathbf{x}}$. The back-propagation step computes gradients of the loss function $\mathcal{L}(\mathbf{x}, \hat{\mathbf{x}})$ with respect to the weights and biases, allowing parameter updates during training (adapted from Nandutu (2023).

2.8 Convolutional Neural Networks

CNNs are a specialized type of NN designed for processing structured grid data, such as images (Montesinos López et al., 2022). They are particularly effective for tasks involving image recognition and classification due to their ability to capture spatial hierarchies in data through convolutional layers. CNNs typically consist of several key layers that work together

to extract features and classify data:

2.8.1 Stride

Stride refers to how far the kernel shifts from one position to the next during a convolution operation. In Figure 2.11 we see two cases; a stride of 1 and a stride of 2. In the first case, the kernel moves one pixel horizontally and vertically across the input matrix. This results in the kernel covering almost every possible region of the input, ensuring that finer details are captured and a larger output feature map is created, as can be seen on the left-hand side of Figure 2.11. On the other hand, if the stride is increased to 2, the kernel moves two pixels at a time, skipping some areas of the input. This results in a smaller output feature map, as fewer parts of the input are scanned. This technique is often used to reduce the size of the input and avoid redundancy.

A smaller stride size such as 1 result in a larger feature map, while a larger stride size such as 2 results in a smaller feature map. The stride size has a direct effect on the resolution of the output. The larger the stride, the more compressed the output. In general, the kernel has a width and height of more than 1, which can lead to output that is smaller than the original input. This often shrinks the image and useful information near the edges of the image may be cut off. To fix this problem and preserve the edge information, padding is applied to the input before folding.

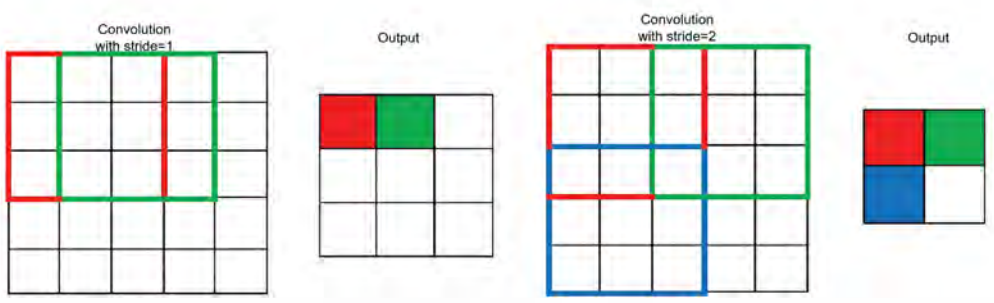


Figure 2.11: A convolution operation with stride sizes of 1 and 2. A stride of 1 ensures finer detail extraction with a larger output feature map, while a stride of 2 results in a more compressed feature map by skipping certain input regions (Vakalopoulou et al., 2023).

2.8.2 Padding

Padding in CNNs refers to the process of adding additional pixels to the edges of an input image before applying the convolution operation. These additional pixels help maintain the spatial dimensions of the input image and ensure that the size of the output feature map remains consistent after multiple convolutions. Padding is typically used to prevent the feature map from shrinking, which can lead to a loss of information at the edges of the image. The most common form of padding is zero padding, where a value of zero is assigned to the

added pixels as shown in Figure 2.12. For example, if the value for zero padding is set to 1, a one-pixel wide border of zeros is added around the image. This allows the convolution filter to glide over more areas of the input, effectively expanding the area in which the filter can process the image.

Padding also plays a crucial role in ensuring that the kernel retains important spatial features, especially near the edges of an image. Without padding, the convolutional filter would reduce the size of the image after each convolution, which could limit the network's ability to detect patterns near the edges.

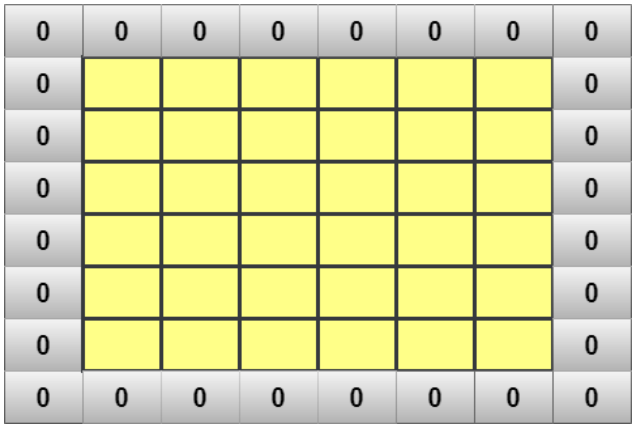


Figure 2.12: A visual representation of zero padding, where a border of zeros is added around the input image to preserve spatial dimensions and retain edge features during the convolution operation.

2.8.3 Convolution Layer

Convolution is a mathematical operation in which two functions are combined to create a third that represents how one function modifies or overlaps the other (Goodfellow et al., 2016). In the context of DL, convolution is used to extract meaningful features from data by applying a filter to an array, such as an image or a sequence. A one dimensional convolution model is expressed as:

$$\begin{aligned} \varphi(t) &= \int_{-\infty}^{\infty} f(\tau)k(t-\tau)\,d\tau \\ &= (f \circ k)(t), \end{aligned} \tag{2.8.1}$$

where $\circ$ represents the convolution operator of the two functions f and k. In image processing, convolution involves overlaying two matrices and the calculation of a weighted sum of the corresponding pixel values. Specifically, CNNs use an input image f and a smaller kernel filter k that slides over the image and analyzes each pixel neighborhood. The resulting output is a feature map, a transformed representation of the original data. The kernel filter effectively resamples the image data and condenses or expands it into a new format. Figure 2.13 shows the basic architecture of CNNs.

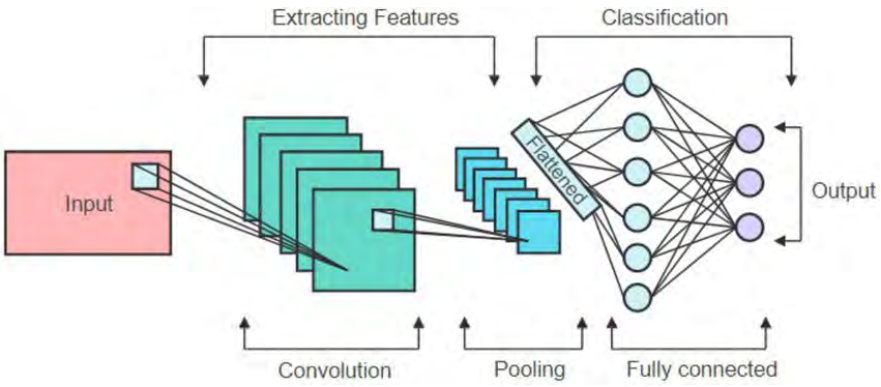


Figure 2.13: A basic architecture of a CNN showing the process of feature extraction through convolution and pooling layers, followed by classification using fully connected layers (Ismail et al., 2023).

This can be extended to a two-dimensional regime, as employed in CNNs:

$$\begin{aligned}\varphi(t_1, t_2) &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(\tau_1, \tau_2) k(t_1 - \tau_1, t_2 - \tau_2)\, d\tau_1\, d\tau_2 \\ &= (f \circ k)(t_1, t_2).\end{aligned} \tag{2.8.2}$$

In the two-dimensional case, convolution is a mathematical operation that combines an input image f and a convolutional kernel k to produce a feature map. This operation involves sliding the kernel across the image and calculating a weighted numerical sum at each position, where the output value represents the response of the kernel to the local region of the image.

In the case of discrete functions (as is typical in digital images), this operation is represented as a summation:

$$\varphi(t_1, t_2) = \sum_m \sum_n f(t_1 + m, t_2 + n) k(m, n), \tag{2.8.3}$$

where m and n represent the offsets from the center of the kernel, and $f(t_1 + m, t_2 + n)$ refers to the pixel values from the input image that are being multiplied by the corresponding values from the kernel $k(m, n)$.

2.8.4 Pooling Layer

Pooling layers are essential in CNNs to reduce the spatial dimensions of feature maps while ensuring that the network remains invariant to small shifts in the input. The two most common pooling operations are max-pooling and average-pooling.

Max-pooling selects the maximum value from a fixed-size window, capturing the most prominent features in a given region. This approach tends to retain only the most important features and discard the less important information. Pooling layers often use a stride equal to the size of the pooling window, which makes it easier to down-sample the feature map. Figure 2.14 shows an example of a max-pooling operation applied to a 4x4 feature map using

a 2x2 filter and a stride of 2.

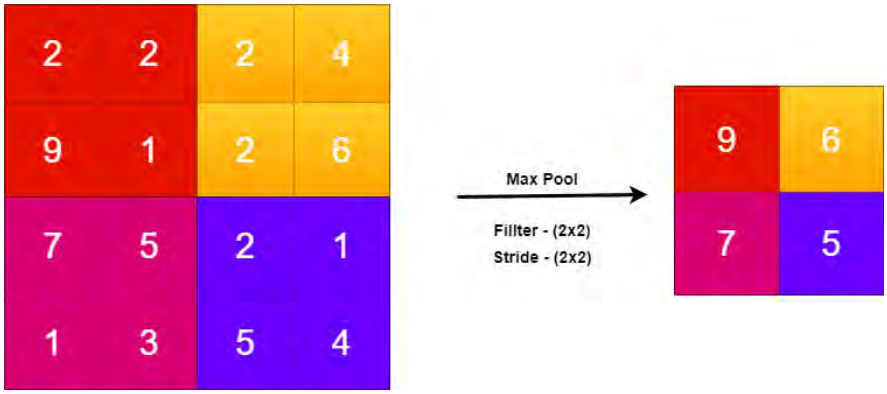


Figure 2.14: An example of a max-pooling operation applied to a 4x4 feature map using a 2x2 filter and a stride of 2. The operation selects the maximum value in each 2x2 region, resulting in a reduced 2x2 feature map that preserves the most prominent features.

In contrast, average pooling calculates the average value of the features within the same window, which allows for a broader representation of the features as all values in the region are taken into account. By combining max-pooling and average-pooling, the network can generalize features more effectively and recognize them regardless of their exact position in the image. This process improves the robustness of the feature extraction mechanism according to the convolutional layers and the associated non-linear activation functions. Figure 2.15 shows an example of an average pooling operation applied to a 4x4 feature map using a 2x2 filter and a stride of 2.

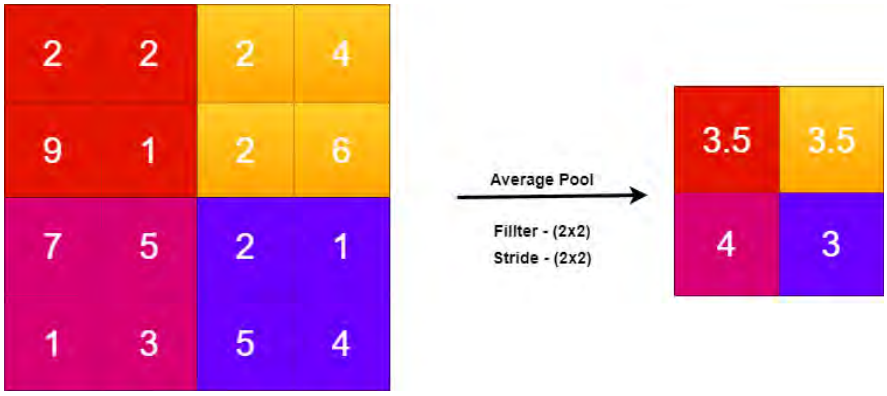


Figure 2.15: An example of average pooling applied to a 4x4 feature map using a 2x2 filter and a stride of 2. The pooling operation computes the average of values within each 2x2 window, resulting in a down-sampled 2x2 feature map that summarizes the original input while retaining spatial information.

2.8.5 Different Architectures of CNNs

CNNs have revolutionized the field of computer vision and image processing. Their ability to automatically and adaptively learn spatial hierarchies of features has made them a critical component of many modern image processing systems. Over the years, various architectures

of CNNs have been proposed, each with their strengths and weaknesses. Below, some of the most popular CNN architectures are discussed.

1. **LeNet-5**

   LeNet-5 is one of the earliest and most influential CNN architectures. Proposed by LeCun et al. (1998), it consists of two convolutional layers followed by two fully connected layers. LeNet-5 was designed for handwritten digit recognition and achieved a high accuracy on the MNIST dataset (Deng, 2012).

2. **AlexNet**

   AlexNet, proposed by Krizhevsky et al. (2012), is a family of CNN architectures that revolutionized computer vision in 2012 by winning the ImageNet Challenge by a wide margin. It extended LeNet by introducing deeper layers, ReLU activations for faster training, and dropout for regularization (Khan et al., 2020). The use of GPUs for training demonstrated the scalability of CNNs on larger datasets and paved the way for modern DL.

3. **VGGNet**

   VGGNet, proposed by Simonyan (2014), is a family of CNN architectures that simplifies design by using small 3x3 convolutional kernels stacked over deeper networks. Although VGG achieved the best performance at its time, its high computational cost and large number of parameters made it clear that more efficient architectures were needed. Despite its inefficiency, VGG's structured design influenced later networks Simonyan (2014).

4. **GoogLeNet**

   GoogLeNet, presented in 2015 by Szegedy et al. (2015) as part of the Inception series, utilized a groundbreaking "inception module" to increase the network's depth and width while maintaining computational efficiency. By processing multiple filter sizes in parallel, it optimized feature abstraction. Subsequent iterations, such as Inception-v3 and Inception-v4, extended this design with innovations such as factorized convolutions and residual connections, achieving improved performance with fewer parameters.

5. **ResNet**

   Residual Networks (ResNets), proposed by He et al. (2016) in 2016, introduced residual connections to solve the vanishing gradient problem and enable the training of very deep NNs. These jump connections enable efficient gradient flow across layers so that networks with hundreds of layers, such as ResNet-50 and ResNet-152, can be trained. Variants such as ResNeXt further improved the architecture by introducing grouped convolutions, striking a balance between performance and computational efficiency

(Hitawala, 2018).

6. **Inception-ResNet**

   The Inception-ResNet proposed in 2016 by Szegedy et al. (2016) is a CNN architecture that combines the strengths of the Inception modules with residual connections. This combination leverages the feature extraction capabilities of Inception modules at multiple levels while improving gradient flow and training efficiency through residual connections. Inception-ResNet introduced deeper and more efficient network designs (Zhang et al., 2018), with variants such as Inception-ResNet-v2 incorporating further optimizations such as factorized convolutions and normalization techniques. These advances have made Inception-ResNet a powerful architecture that allows you to achieve high performance on complex image recognition tasks.

7. **DenseNet**

   DenseNet, proposed by Huang et al. (2017) in 2017 and inspired by ResNet, introduces dense connections that link each layer to all subsequent layers. This design promotes feature reuse, improves information flow and reduces the number of parameters compared to traditional architectures. Unlike ResNet, which focuses on bypassing information via skip connections, DenseNet ensures that each layer contributes directly to subsequent layers, increasing efficiency in both learning and inference. Variants such as DenseNet-201 are an example of the architecture's ability to achieve high performance while maintaining computational efficiency.

8. **MobileNet**

   MobileNet, proposed by Howard (2017) in 2017, is a CNN architecture designed for mobile and embedded systems. It uses depth-wise separable convolutions to significantly reduce computational costs and memory requirements while maintaining performance. Subsequent variants, such as MobileNetV2, introduced innovations such as inverted residuals and linear bottlenecks to further improve the efficiency and accuracy of the architecture (Sandler et al., 2018). These advances make MobileNet and its variants ideal for resource-constrained environments where computational efficiency is critical.

2.9 Unsupervised Machine learning

In unsupervised ML, algorithms are trained on data without labelled outputs with the aim of discovering patterns, structures or relationships in the data. Unsupervised learning involves training on data $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$ without any associated labels, with the goal of finding patterns, clusters, or anomalies in the data. A common task is clustering, where the goal is to partition X into K clusters. The objective of clustering algorithms such as K-Means is to

minimise the within-cluster sum of squares:

$$\min_{C_1,\ldots,C_K} \sum_{k=1}^{K} \sum_{\mathbf{x}_i \in C_k} \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2, \tag{2.9.1}$$

where C_k represents the k-th cluster and $\boldsymbol{\mu}_k$ is the centroid of that cluster.

Below is an overview of unsupervised learning models suitable for this work.

2.9.1 Autoencoders

AEs are a powerful class of unsupervised NNs that have attracted considerable attention in the field of anomaly detection, mainly due to their ability to learn compressed, meaningful representations of input data. They are widely applied in tasks such as feature extraction, dimensionality reduction, data compression, and anomaly detection. A core concept in AEs is dimensionality reduction, where high-dimensional input data is transformed into a low-dimensional latent representation, often referred to as a latent code (Hinton & Salakhutdinov, 2006). An AE usually consists of three main components: an encoder, a decoder, and a latent space, also known as a bottleneck (see Figure 2.16).

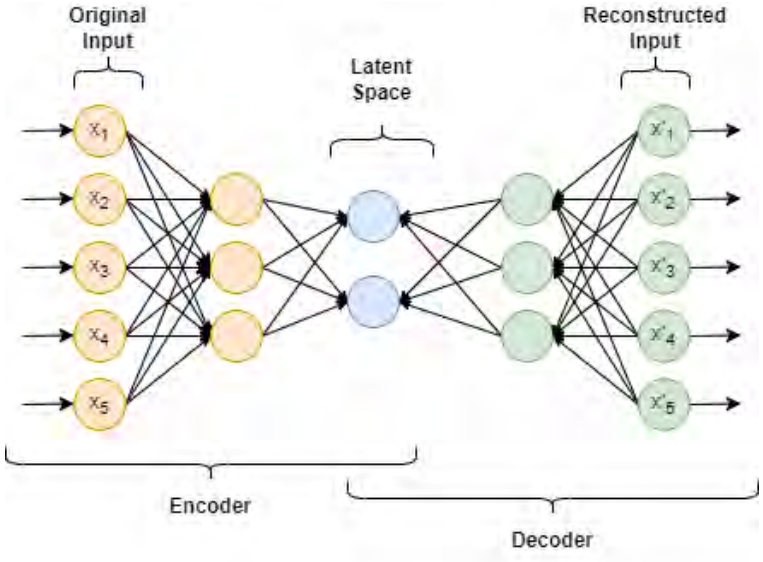


Figure 2.16: A detailed architecture of an autoencoder (AE) for anomaly detection in telemetry datasets, comprising original input data (in orange), latent space (also called a bottleneck, in blue), and reconstructed data (in green).

The architecture of an autoencoder is composed of three main components. The first component is the encoder, which aims to transform the input data $\mathbf{x}$ into a compressed representation called the latent space $\mathbf{z}$. It is composed of neural layers that gradually reduce the dimensionality of the input data. Mathematically, the encoder can be expressed as:

$$\mathbf{z} = \gamma_1(\mathbf{w}_i\mathbf{x} + \mathbf{b}_i), \tag{2.9.2}$$

where γ_1 is the activation function for the encoder, $\mathbf{w}_i$ represents the weight matrix for the i-th neuron in the encoder layer, $\mathbf{b}_i$ is the bias vector for the i-th neuron in the encoder layer, and $\mathbf{z}$ represents the output of the encoder. The second component of the autoencoder is the latent space, also known as the bottleneck. This latent space is a reduced-dimensional form of the input data as produced by the encoder. The size of the latent space, defined by the number of neurons in the bottleneck layer, is a crucial hyperparameter that can be adjusted depending on the specific application and the desired level of compression.

The final component of the autoencoder is the decoder. The role of the decoder is to take the compressed representation from the encoder and reconstruct the original input data. The decoder consists of one or more neural layers that progressively increase the dimensionality of the latent representation until it matches the original input. The output layer of the decoder should have the same shape as the input data and use an appropriate activation function for the data type. Mathematically, the decoder can be expressed as:

$$\mathbf{x}' = \gamma_2(\mathbf{w}_j\mathbf{z} + \mathbf{b}_j), \tag{2.9.3}$$

where γ_2 is the activation function for the decoder, $\mathbf{w}_j$ represents the weight matrix for the j-th neuron in the decoder layer, $\mathbf{b}_j$ is the bias vector for the j-th neuron in the decoder layer, and $\mathbf{x}'_j$ represents the reconstructed output data from the j-th neuron.

In the context of anomaly detection, the autoencoder is trained on normal instances only so that it can learn the underlying patterns and structures of the data. If the model encounters anomalous data during inference it is expected to perform poorly during reconstruction, resulting in a higher reconstruction error. This deviation can be used to identify anomalies. Based on the reconstruction error, a threshold is set above which data points are classified as anomalies. The choice of this threshold is crucial and often depends on the distribution of reconstruction errors observed during the training phase (Berahmand et al., 2024; Ahmed et al., 2016).

2.9.2 Isolation Forest

IF, introduced by Liu et al. (2008), is an algorithm for detecting anomalies in datasets. At the heart of the IF algorithm is the concept of the isolation tree. An isolation tree is a binary tree constructed recursively aiming to isolate individual data points. It operates on the premise that anomalies are easier to isolate from the majority of data points, making them stand out as they require fewer splits in a DT (Figure 2.17). The process begins by selecting a random feature and a random value within the range of that feature. The dataset is then split into two subsets: one containing data points with values below the chosen value, and the other containing points with values above it. This partitioning continues recursively until a single data point remains isolated or a predefined tree depth is reached. In Figure 2.17, anomalies (red) are identified as leaf nodes with shallow depths (short path lengths), while

normal data points (blue) are classified as those located in leaf nodes with greater depths (long path lengths). While data points shown in yellow are uncertain, this might be because they have path lengths (depths) that are similar to those of both normal and anomalous data points. The path length for a data point $\mathbf{x}$ in the isolation tree can be expressed as a recursive function:

$$h(\mathbf{x_i}) = \begin{cases} 0 & \text{if } x \text{ is a leaf} \\ h(\mathbf{x}_{\text{left}}) + 1 & \text{if } x \text{ has a left child} \\ h(\mathbf{x}_{\text{right}}) + 1 & \text{if } x \text{ has a right child.} \end{cases} \tag{2.9.4}$$

Here, $\mathbf{x}_{\text{left}}$ and $\mathbf{x}_{\text{right}}$ denote the left and right child nodes of $\mathbf{x}_{\text{i}}$ in the binary isolation tree. This helps quantify how easy it is to isolate a particular point, with anomalies having shorter path lengths due to their rarity.

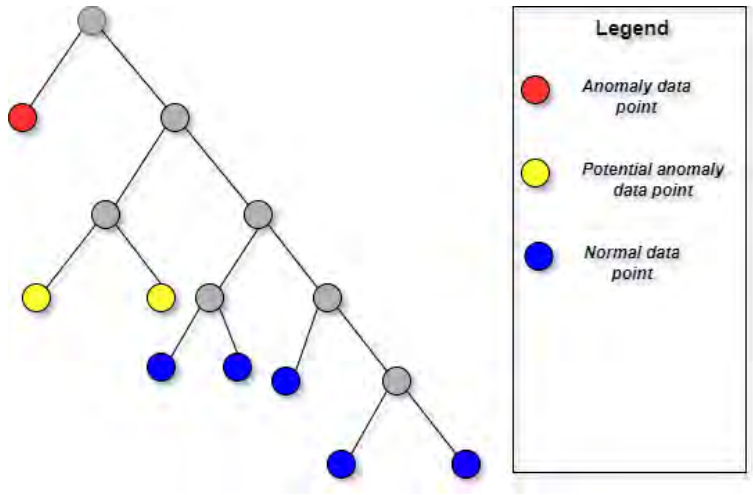


Figure 2.17: Isolation tree illustrating anomaly detection. This figure shows a binary isolation tree constructed to detect anomalies in a dataset. Anomalies are shown as red points, normal data points in blue, and uncertain points, which are similar to both normal and anomalous points, in yellow.

Finally, the anomalies are defined based on a scoring mechanism (anomaly score) that quantifies their level of isolation. This scoring mechanism is crucial for identifying anomalies in the IF algorithm. The anomaly score for a data point $\mathbf{x}$ is calculated as:

$$s(\mathbf{x}) = 2^{-\frac{E(h(\mathbf{x},T_i))}{c(T_i)}}, \tag{2.9.5}$$

where T_i represents an individual isolation tree and i ranges from 1 to the number of trees built, $h(\mathbf{x}, T_i)$ is the path length of data point $\mathbf{x}$ in tree T_i. $c(T_i)$ is the average path length of all data points in tree T_i, $E(h(\mathbf{x}, T_i))$ is the expected path length of x in tree T_i, calculated based on the average path length $c(T_i)$ of the tree. $s(\mathbf{x})$ is the anomaly score of data point $\mathbf{x}$, a score close to 1 indicates an anomaly (i.e., shorter path lengths), while a score closer to 0 suggests a normal point with longer path lengths.

In the IF algorithm, the path length of a data point serves as the basis for calculating its anomaly score. The path length represents the number of edges or splits that must be traversed within the tree to isolate the data point. According to Liu et al. (2008), anomalies tend to have shorter path lengths in the resulting trees, requiring fewer partitions to be isolated compared to normal data points (Figure 2.18).

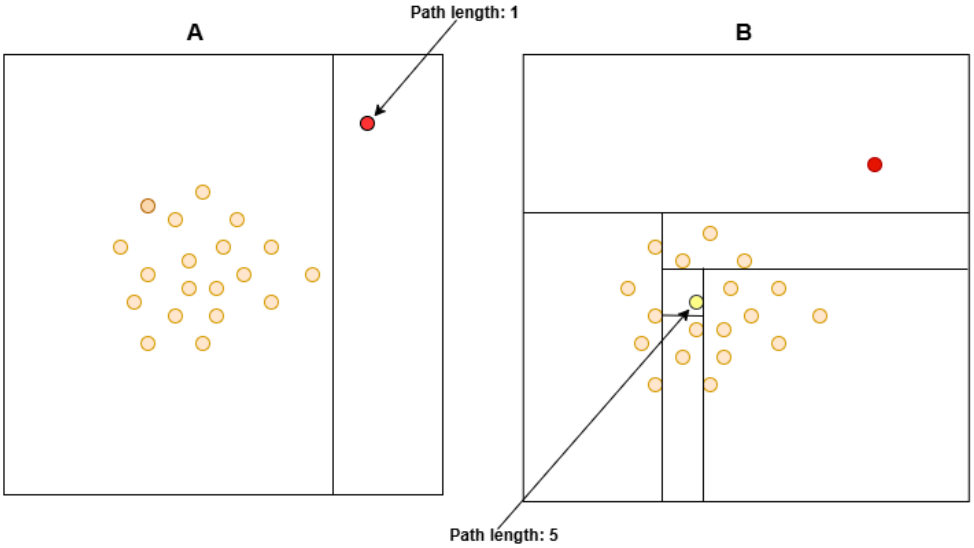


Figure 2.18: Path length comparison for anomalies and normal data points. This figure shows path length differences in an isolation tree. Anomalies (red) are isolated with fewer partitions (A), while normal data points (yellow) require more (B).

To further optimize the IF algorithm, we focus on the gradient of the anomaly score $s(\mathbf{x})$ to improve tree construction efficiency:

$$\frac{\partial s(x)}{\partial x} = -\frac{\ln(2)}{c(N)} 2^{-\frac{E(h(x))}{c(N)}} \frac{\partial E(h(x))}{\partial x}, \tag{2.9.6}$$

where $c(N)$ is the average path length for a dataset of size N, approximated as:

$$c(N) = 2H(N-1) - \frac{2(N-1)}{N} \tag{2.9.7}$$

and $H(i)$ is the i-th harmonic number. This approach uses gradients to improve the scoring system, allowing for faster and more accurate detection of anomalies.

2.9.3 Local Outlier Factor

LOF is a widely used unsupervised method for detecting anomalies. It identifies outliers by assessing the local density of a data point relative to its neighbors (Breunig et al., 2000), rather than relying solely on the global distribution of the data. This approach is particularly effective for datasets with significant density variation across different regions. The LOF

algorithm works by calculating the local reachability density for each data point and then comparing it to the local reachability densities of its neighbors.

To apply the LOF method, the first step involves using the Euclidean distance to determine the k-nearest neighbors (k-NN) and the reachability distance for each data point. The value of k is used to identify the closest points in the dataset, which form the k-nearest neighborhood. Once the neighborhood is established, the local reachability density (LRD_k) for a given point p is calculated using the reachability distance between p and its neighbors. The reachability distance between p and a neighbor o is defined as:

$$\mathrm{reach\_dist}_k(p, o) = \max\left\{\text{k-distance}(o), \text{distance}(p, o)\right\}, \tag{2.9.8}$$

where distance(p, o) is the Euclidean distance between p and o, and k-distance(o) is the distance from o to its k-th nearest neighbor. This ensures stability in density estimates by preventing artificially small distances.

The local reachability density is expressed as:

$$\mathrm{LRD}_k(p) = \frac{\sum_{o \in N_k(p)} \mathrm{reach\_dist}_k(p, o)}{|N_k(p)|}, \tag{2.9.9}$$

where $\mathrm{reach\_dist}_k(p, o)$ is the reachability distance of a point p with respect to another point o. Here, $N_k(p)$ denotes the set of k-nearest neighbors of p. This distance is defined as the maximum value between the k-distance of o and the actual distance between p and o.

Once the local reachability densities are obtained, the LOF score (LOF_k) for each data point p is computed. The LOF score measures how anomalous a point p is relative to its local neighborhood by comparing its local density to those of its neighbors. Specifically, the LOF score is calculated as the average ratio of the local reachability density of p to that of its k-nearest neighbors:

$$\mathrm{LOF}_k(p) = \frac{|N_k(p)|}{\sum_{o \in N_k(p)} \mathrm{LRD}_k(p)\mathrm{LRD}_k(o)}. \tag{2.9.10}$$

A higher LOF score indicates that p is an outlier relative to its local neighborhood, as its density significantly deviates from the densities of its surrounding points.

To enhance the efficiency of the LOF algorithm, optimization techniques are applied to improve the computation of the density measures. One such technique involves calculating the gradient of the reachability distance with respect to the data points, ensuring that LOF scores are updated efficiently. The gradient of the local reachability density (LRD_k) for a data point p is expressed as:

$$\frac{\partial \mathrm{LRD}_k(p)}{\partial p} = -\left(\frac{\sum_{o \in N_k(p)} \mathrm{reach\_dist}_k(o, p)}{|N_k(p)|}\right)^{-2} \frac{\partial}{\partial p}\left(\frac{\sum_{o \in N_k(p)} \mathrm{reach\_dist}_k(o, p)}{|N_k(p)|}\right), \tag{2.9.11}$$

where the partial derivatives ensure that the local densities are efficiently updated as the model iterates. By leveraging such optimization techniques, the LOF algorithm achieves improved performance in identifying outliers, especially in large datasets with varying densities.

2.9.4 Density-based spatial clustering of application with noise (DBSCAN)

DBSCAN is a fundamental clustering algorithm in the field of ML, known for its ability to identify clusters of varying shapes and sizes while effectively handling noise. The algorithm groups data points located in dense regions, distinguishing them from outliers or noise points that are sparsely distributed (Ester et al., 1996). DBSCAN relies on two key parameters: the *epsilon* (ϵ), which defines the maximum distance between two points to consider them neighbors, and the minimum number of points (*MinPts*) required to form a dense region. Typical values for *MinPts* range between 3 and 10, depending on the characteristics of the dataset. The concept of clusters in DBSCAN is built around core points, border points, and noise points. Figure 2.19 illustrates how clusters are formed based on density, highlighting core points (green), border points (orange), and noise points (red).

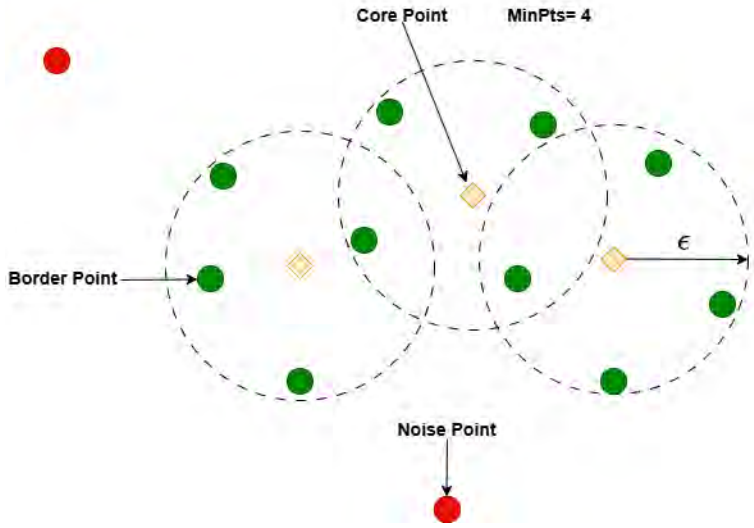


Figure 2.19: This DBSCAN diagram shows the clustering process where core points (green) are surrounded by a minimum number of points ($MinPts = 4$) within a radius ϵ. It distinguishes between core points, border points (orange), and noise points (red). The diagram illustrates how clusters are formed based on density, with core points expanding the cluster by including all reachable points within ϵ (adapted from Zhu et al. (2023)).

The DBSCAN algorithm begins by selecting its parameters, ϵ and *MinPts*, which control the clustering process. A data point is identified as a *core point* if it has at least *MinPts* neighboring points within the distance ϵ. Core points serve as the starting points for clusters. From each core point, DBSCAN identifies all points that are *density-reachable*, meaning they can be reached directly or indirectly from the core point within the specified distance ϵ.

A point q is *density-reachable* from p if there exists a path of points where each is within distance ϵ of the next, starting from p and ending at q. These reachable points are iteratively added to the cluster as the algorithm expands outward. The process continues until no new points can be included in the cluster, ensuring the formation of dense regions. Points that do not meet the core point criterion and have insufficient neighboring points within ϵ are classified as *noise points* or outliers.

To enhance the efficiency of DBSCAN, optimization techniques focus on calculating core points and density-reachable points using gradients. Specifically, the gradient of the density-reachability with respect to the distance metric ϵ is expressed as:

$$\frac{\partial \text{reach\_dist}_\epsilon(p,q)}{\partial \epsilon} = \delta(p,q), \tag{2.9.12}$$

where $\text{reach\_dist}_\epsilon(p,q)$ is the reachability distance between points p and q, and $\delta(p,q)$ is an indicator function where $\delta(p,q) = 1$ if q is density-reachable from p under ϵ, else 0. This gradient-based optimization accelerates the clustering process by dynamically adjusting ϵ, thereby improving the overall efficiency of the algorithm.

2.9.5 Long Short-Term Memory

LSTM is a specific Recurrent Neural Network (RNN) architecture designed to address the limitations encountered by traditional RNNs when capturing long-term dependencies in sequential data (Ahsan & Salah, 2023). The key innovation behind LSTMs lies in the introduction of a memory cell, which acts as a conveyor belt, enabling information to flow through time steps with minimal delay. This structure is complemented by gating mechanisms that allow the network to selectively retain or discard information. These properties make LSTMs particularly effective for tasks requiring the modeling of long-term dependencies, such as natural language processing, time series forecasting, and speech recognition. Figure 2.20 illustrates the architecture of a typical LSTM block, which consists of four key components: the input gate, forget gate, output gate, and memory cell.

The central element of the LSTM architecture is the *cell state* C_t, which represents the memory of the network and carries information across time steps. It is updated at each step using the forget gate and the input gate, as follows:

$$C_t = f_t C_{t-1} + i_t \cdot \tilde{C}_t. \tag{2.9.13}$$

Here, the forget gate f_t determines which information from the previous cell state should be discarded. It takes the previous hidden state h_{t-1} and the current input x_t as inputs and uses a sigmoid activation function σ to produce values between 0 and 1:

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f). \tag{2.9.14}$$

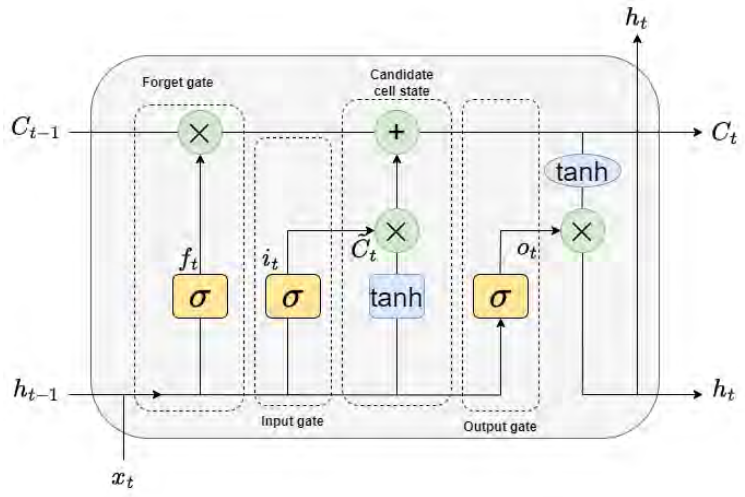


Figure 2.20: The neural unit structure of an LSTM network. The block consists of four main components: an input gate, a forget gate, an output gate, and a memory cell. These components work together to help the network learn and retain complex temporal relationships in data (adapted from Elgindy et al. (2023)).

The input gate i_t decides which new information is added to the cell state. It processes the concatenated vector $[h_{t-1}, x_t]$ (which combines the previous hidden state and the current input) using a sigmoid activation:

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i). \tag{2.9.15}$$

The candidate cell state $\tilde{C}_t$ computes new candidate values for the memory cell using a tanh activation function. The tanh activation ensures candidate values are normalized to [-1, 1], stabilizing gradient flow:

$$\tilde{C}_t = \tanh(W_c[h_{t-1}, x_t] + b_c). \tag{2.9.16}$$

The hidden state h_t is the output of the LSTM at each time step, representing the relevant information carried forward to the next step. It is computed as:

$$h_t = O_t \tanh(C_t), \tag{2.9.17}$$

where O_t is the output gate, responsible for regulating the flow of information from the cell state. It uses the sigmoid activation function:

$$O_t = \sigma(W_o[h_{t-1}, x_t] + b_o). \tag{2.9.18}$$

In the equations above, W_f, W_i, W_c, and W_o represent the weight matrices associated with the forget gate, input gate, candidate cell state, and output gate, respectively. Similarly, b_f, b_i, b_c, and b_o denote the corresponding bias vectors. These parameters are learned during training and are applied to the concatenated vector of the previous hidden state h_{t-1} and the

current input x_t to control the gate activations.

These gates collectively enable LSTMs to iteratively refine their memory and focus, making them capable of retaining relevant information over long sequences. The interplay between short-term and long-term dependencies allows LSTMs to handle complex temporal relationships effectively. As a result, LSTMs are widely applied in sequential tasks such as natural language processing, time series analysis, financial forecasting, and speech recognition, where the ability to identify long-term patterns is crucial for accurate predictions and interpretations.

2.10 Reinforcement Learning

RL is a type of ML in which an agent learns to make decisions by interacting with an environment and receiving feedback in the form of rewards or penalties (Kaelbling et al., 1996). The agent aims to maximize cumulative rewards by learning a strategy that guides its actions based on the state of the environment. Figure 2.21 depicts the RL framework, where the agent interacts with the environment by performing actions, observes the resulting state, and receives feedback in the form of rewards or penalties.

In contrast to supervised and unsupervised learning, RL involves trial and error and focuses on sequential decision making. Common RL algorithms include Q-learning and Deep Q-Networks (DQN). RL has played an important role in various areas: In autonomous vehicles, agents learn to navigate traffic, avoid collisions and make efficient routing decisions (Xiang, 2024); in robotics, RL enables robots to grasp objects, walk and assemble components through iterative learning (Sekkat et al., 2024); in healthcare, it optimizes treatment plans and accelerates drug discovery by exploring vast chemical spaces (Al-Hamadani et al., 2024); and in energy management, RL reduces operating costs by optimizing resource allocation in smart grids and cooling systems (Rayati et al., 2015).

By adapting through trial and error, RL provides a powerful framework for solving problems that require long-term planning and goal-oriented optimization, making it a key part of modern AI advancements.

2.11 Summary

This chapter has explored the basic principles and advanced techniques of ML and DL, beginning with an overview of ML paradigms (supervised and unsupervised). The core structures such as perceptrons which form the basis for more complex architectures such as ANNs, were then discussed.

The training processes underlying these models were investigated, including the activation functions, feed-forward process and back-propagation, which are essential for optimizing the

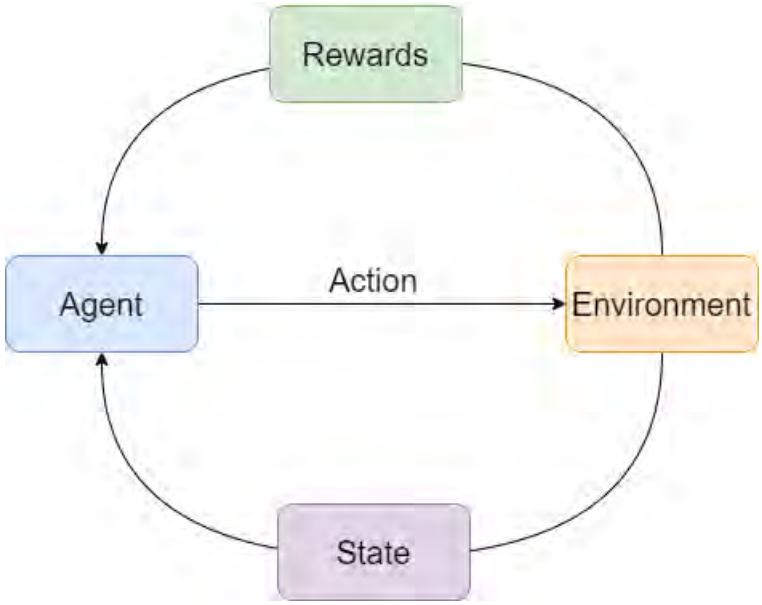


Figure 2.21: The reinforcement learning loop, where an agent interacts with the environment by taking actions. The environment responds by updating the state and providing rewards, guiding the agent's actions to maximize performance.

performance of neural networks. Subsequently, deep learning architectures in particular CNNs were examined for their effectiveness in recognizing patterns in structured data such as images or, in this case, acoustic telemetry data. Special attention was paid to CNN components such as convolution, stride and padding, which are crucial for understanding how the network processes and extracts features from the input data. Several well-known CNN architectures were also discussed.

In addition, we discussed advanced models such as LSTM networks and AEs, which are particularly suitable for capturing temporal dependencies and compressing data for anomaly detection. Unsupervised algorithms such as IF and LOF were also investigated, which offer alternative approaches for anomaly detection without labelled datasets.

The understanding gained in this chapter formed the theoretical basis for the implementation of these algorithms in the context of acoustic telemetry data. These methods were applied in the following chapters to detect anomalies in large datasets of dusky kob *Argyrosomus japonicus* movements in the Breede Estuary, South Africa.

Chapter 3

Telemetry Dataset

This chapter introduces the telemetry dataset used in this work, focusing on dusky kob *Argyrosomus japonicus* acoustically tagged in the Breede Estuary. The tagging and data collection processes are described in detail in Section 3.2. Sections 3.3 to 3.6 describe the pre-processing, standardization and feature engineering steps that were applied to the raw data, including the labelling process for anomaly detection. Section 3.7 discusses the generation of synthetic telemetry data.

3.1 Study site and species

The Breede Estuary (Figure 3.1) is 52 km in length and is a vital ecological and hydrological system that flows into the Indian Ocean at the coastal town of Witsand (Ziko et al., 2023). The Breede Estuary features diverse habitats like mudflats, channels, and intertidal zones, supporting a wide range of aquatic species, including several fish species of recreational and small-scale/subsistence importance Childs et al. (2015), such as spotted grunter *Pomadasys commersonnii*, leervis *Lichia amia* and dusky kob (Cowley et al., 2006).

The dusky kob is an important coastal fishery species with a South African distribution along the southeast coast from Cape Point to southern Mozambique (more abundant from Cape Agulhas to northern KwaZulu-Natal) (Griffiths & Heemstra, 1995). Its life history is well known, with juveniles being dependent on estuaries as nursery habitats (Griffiths, 1996; Ter Morhuizen et al., 1996), and even large adults remaining in estuaries for extended periods of time (up to seven or more years; Acoustic Tracking Array Platform (ATAP), unpublished data). Their movements in South Africa have been well studied using acoustic telemetry (Cowley et al., 2008; Næsje et al., 2012; Childs, 2013; Childs et al., 2015). Juveniles are highly resident to estuaries (Cowley et al., 2008; Childs et al., 2015), and as adults (ATAP, unpublished data), making extensive use of these systems while in them (Næsje et al., 2012). Given the long-term residency to estuaries, and their wide-scale use of most estuaries in which

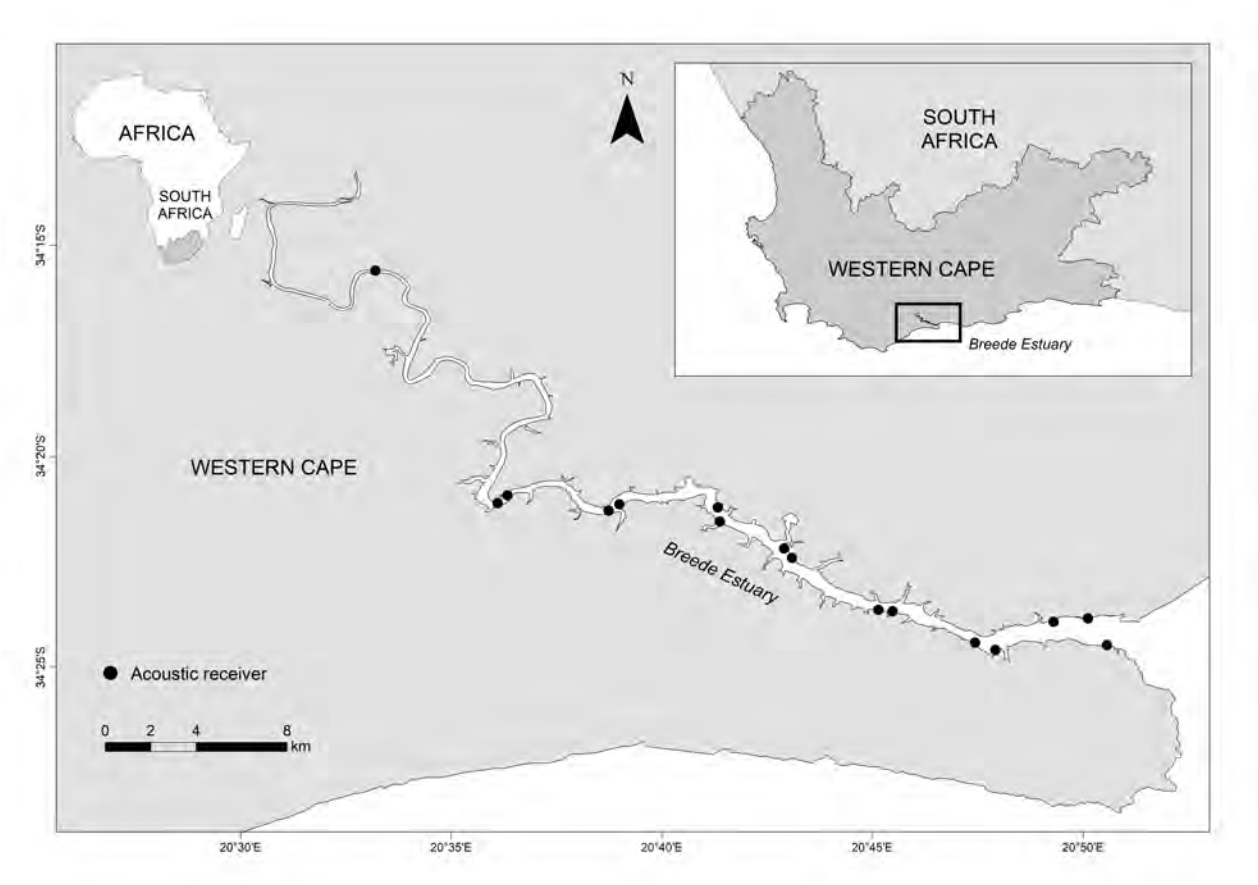


Figure 3.1: Map of the Breede Estuary showing the locations of acoustic receivers (black pins) deployed to monitor dusky kob movements in the estuary between 2016 and 2021.

they occur (i.e. lower to upper reaches), acoustic telemetry data of dusky kob collected in estuaries form the ideal time series dataset for ML and DL model training.

3.2 Tagging and data collection

Between 2016 and 2021, 50 dusky kob, ranging in length from 660 to 1720 mm Total Length (TL) (average ± SD: 1196 ± 279 mm TL) were caught and tagged with individually coded acoustic transmitters (V13 and V16 transmitters, 69 kHz, Innovasea, Canada) (Table 3.1). The tags used in this study were coded acoustic transmitters set with a 60 to 180 s nominal delay. As such, they emitted an acoustic signal anywhere between 60 and 180 s; however, the time between emissions is not set at a specific interval. The manufacturer (Innovasea) programmes tags in this way to reduce the probability of code collisions. These code collisions i.e. when two acoustic signals are detected by an acoustic receiver at the same time, can result in the receiver not knowing how to interpret the signals. As such, a false detection can be recorded where the signal is interpreted as an existing tag ID, or it can be recorded as an entirely new and unusual tag ID.

The tagging process involved catching fish using conventional rod and line techniques from

a small boat or the bank of the estuary. Once caught, the fish was carefully pulled onto the boat, and placed onto a large V-shaped high-density foam cradle and prepared to surgery. The head was covered with a cloth to reduce stress, and water was continually flushed over the gills throughout the surgical procedure. The surgical procedure involved making a small incision (approximately 1.5 to 2 cm) in the fish's abdomen between the pelvic and anal fin, inserting the transmitter into the coelomic cavity, and then stitching the incision with two or three sutures. Once sutured, an antibacterial wound powder was placed over the incision site which congealed upon wetting. The surgical procedure followed that of Cowley et al. (2008). Ethical approval was obtained from the South African Institute for Aquatic Biodiversity's Animal Ethics Committee (approval no. 2013_06), and research permits were obtained from the Department of Forestry, Fisheries and the Environment (permit no. RES2017/26, RES2019/16, RES2020/45).

The movements of the tagged dusky kob were monitored in the Breede Estuary between 2016 and 2021 using an array of 16 passive acoustic receivers (model VR2W, 69 kHz, Innovasea, Canada) deployed throughout the estuary (Figure 3.1). The effective detection range of these receivers varied between 110 and 610 meters depending on environmental conditions such as salinity, temperature, wave action, and physical obstructions (e.g., bridge pylons or rocks), with an estimated range of $\sim$ 500 meters in the Breede Estuary under typical conditions (Kerwath et al., 2005; Næsje et al., 2012; Grant et al., 2017). When a tagged fish was detected, its location was inferred to be within this detection range, consistent with standard practices in estuarine telemetry studies (Hindell et al., 2008; Sakabe & Lyle, 2010). Receivers were downloaded once a year to secure data using Innovasea's VUE software.

Overall, all 50 kob were detected at least once, together accumulating 3013930 detections (Table 3.1). The average number of detections recorded per individual was 60278 ($\pm$ 63276), ranging from 23 to 259676 detections, and individuals were detected for an average of 184 ($\pm$ 217) days throughout the monitoring period, ranging from 2 to 1061 days (Table 3.1). The average number of detections recorded per day ranged from 11 to 854 average $\pm$ SD: 336 $\pm$ 128).

Table 3.1: Summary of tagging and acoustic monitoring details for dusky kob (*Argyrosomus japonicus*) in the Breede Estuary (2016-2021), showing total length, tagging date, detection count, station, and monitoring duration for each individual.

FishID	Total length (mm TL)	Tag date	No. of detections	No. of unique stations visited	No. of days detected
A69-9001-23701	1280	2015-11-01	71148	15	194
A69-9001-23702	990	2016-01-03	23	1	2
A69-9001-23668	1450	2016-10-05	42408	15	80
A69-9001-23669	1600	2016-10-15	67564	16	164
A69-9001-23670	1280	2016-10-16	88973	15	258
A69-9001-23671	1380	2016-10-16	76810	16	226
A69-9001-23672	1400	2016-10-17	8927	14	25
A69-9001-23728	1210	2016-10-17	73156	16	250
A69-9001-23727	1210	2016-10-18	12865	15	53
A69-9001-23726	1460	2016-10-18	27984	15	76
A69-9001-23737	1510	2016-10-18	84419	15	249
A69-9001-23738	1400	2016-10-18	7905	14	24
A69-9001-23739	1390	2016-10-18	57754	16	188
A69-9001-23747	1460	2016-10-18	86081	16	208
A69-9001-23745	1180	2016-10-19	33178	16	134
A69-9001-23746	1380	2016-10-19	25535	15	60
A69-9001-23748a	1140	2016-10-19	39614	15	123
A69-9001-23749	1290	2016-10-19	37648	15	100
A69-9001-23750	1239	2016-10-19	112064	16	276
A69-9001-23751	1270	2016-10-19	57073	15	137
A69-9001-23754a	1180	2016-10-20	14804	15	63
A69-9001-23754b	705	2018-08-10	422	3	23
A69-9001-23675b	750	2018-08-10	149205	15	542
A69-9001-23766b	1140	2016-10-20	55565	16	150
A69-9001-23753	1340	2016-10-20	29121	16	100
A69-9001-23697b	1230	2016-10-20	50402	15	122

Table 3.1: Cont.

FishID	Total length (mm TL)	Tag date	No. of detections	No. of unique stations visited	No. of days detected
A69-9001-23698b	1210	2016-10-29	41306	15	112
A69-9001-23682b	1140	2016-10-29	15749	15	45
A69-9001-50373b	1290	2016-10-29	10480	15	54
A69-9001-23774	1490	2016-10-29	13504	12	28
A69-9001-23741b	1310	2016-10-29	80670	14	212
A69-9001-23721	1230	2016-11-02	56559	16	193
A69-9001-23722	1190	2016-11-02	259676	1	1061
A69-9001-23723a	1340	2016-11-02	9837	15	34
A69-9001-23777	1190	2016-11-02	15067	13	54
A69-9001-23778	1190	2016-11-02	29839	16	126
A69-9001-23779	1160	2016-11-02	4461	11	18
A69-9001-23775a	1240	2016-11-02	57840	16	216
A69-9001-23724	1460	2016-11-11	2567	11	10
A69-9001-23732	1720	2017-04-12	127797	16	247
A69-9001-23769	1650	2017-05-05	4271	3	5
A69-9001-23730	1220	2017-02-24	8094	15	31
A69-9001-23858a	695	2017-08-25	239874	16	461
A69-9001-23859a	660	2017-08-25	17679	15	63
A69-9001-23860a	665	2017-08-26	214246	15	561
A69-9001-23861a	660	2017-08-30	44924	10	135
A69-9001-23862	680	2017-09-19	225462	16	953
A69-9001-23714a	700	2017-11-28	16364	9	37
A69-9001-23684b	695	2018-03-29	96051	15	484
A69-9001-23733	665	2018-09-20	110965	14	251

3.3 Data pre-processing

Before training the ML and DL classifiers, several important pre-processing steps were carried out to ensure the integrity and reliability of the telemetry data. One of the first tasks was to deal with missing values. Missing values in acoustic telemetry refer to gaps due to technical problems (e.g. receiver failure), instances where no detection were recorded at expected timestamps or locations, or where some features within the detection records were incomplete or missing. All such records with missing values were removed to maintain data consistency. Duplicates were also removed to optimise computational resources, as retaining multiple identical detections unnecessarily increases the size of the dataset and the processing time. In the context of acoustic telemetry, duplicates typically refer to multiple identical detections of the same fish by the same receiver at the same timestamp. These can result from errors in data collection or transmission and artificially increase the number of detections, which can lead to misleading interpretations of fish presence.

Another critical point was the occurrence of colliding signals that led to the creation of false fish IDs. These false detections usually resulted from overlapping transmissions between acoustic tags, which led to false identifications by the receivers. Removing these false IDs was essential for maintaining the biological credibility of the dataset. In addition, we standardized the data to ensure consistency between all detection stations.

Once the data was cleaned and standardized, it was split into training, validation, and test sets to facilitate the model development process. This split allowed for robust training, model tuning, and unbiased evaluation of performance. Each of these pre-processing steps played a crucial role in ensuring that the data used for model training accurately reflected fish movements and avoided any biases that could skew the ability of ML models to effectively detect anomalies (Marciuc, 2021).

3.4 Data Standardization

In this work, one of the crucial steps before applying anomaly detection methods was to standardize the data. Standardization was applied exclusively to all numerical features because categorical variables such as fish IDs and station names do not require scaling and numerical features with different units require comparable ranges for accurate model training. The goal of standardization is to bring all features or variables to the same scale to ensure that they are treated equally by the ML algorithms. This process ensured that each dataset was transformed to have a mean (μ) of zero and a standard deviation (σ) of one (Ismael, 2025). For a given dataset D, where μ is the mean and σ is the standard deviation, the transformation of each data point x is performed using the following formula:

$$\hat{x} = \frac{x - \mu}{\sigma}, \quad \forall x \in D. \tag{3.4.1}$$

The standardization of the data, as described in the equation above, plays a crucial role in improving the performance of the ML models used in this work. This is especially beneficial for multivariate time series data with features that exist at different scales. By standardizing the data, this project accelerate the convergence of many ML algorithms (Shanker et al., 1996). For the methods used in this work, such as NN-AE, DBSCAN, IF, and LOF, the study found performance improvements when the data were standardized, making this an essential pre-processing step for effective anomaly detection.

3.5 Engineered features

In addition to pre-processing, several new features were engineered to enhance the anomaly detection process. Engineered features play a crucial role in ML and DL models, as they help capture details that are not directly present in the raw data, ultimately improving the performance of the model. In this case, these included "duration at the same station", "number of detections", "number of days detected", "number of unique stations", and "consecutive missing stations" (Table 3.3).

One of the most important features was "num unique stations", which captures how many different stations a fish was detected on in a given time period. This feature was crucial for assessing the spatial extent of fish movements, as a fish that constantly moved between multiple stations was considered normal, while deviations such as irregular or limited station movements indicated anomalies. Another important feature was "num detections", which recorded the total number of detections of a fish and provides information on the general level of activity of the fish and habitat use.

The feature "duration in same station", also referred to as length of stay at a station, measured how long a fish stayed at a single station. Longer stays at a single station can indicate either tag malfunction or uncharacteristic behaviour, making this feature particularly useful for identifying potential anomalies. Similarly, the "avg time between detections" feature was introduced to track the time between successive detections at different stations, with unusually fast or slow transitions serving as key indicators of abnormal movements.

Additionally, the feature "distance" was introduced to quantify the spatial movement, which was previously unspecified. The distance feature, for example, is calculated using the Haversine formula to determine the great-circle distance between two geographic points. According to Santhosh & Idicula (2017), the Haversine formula is defined as follows:

$$d = 2r \cdot \arcsin \sqrt{\sin^2\left(\frac{\varphi_B - \varphi_A}{2}\right) + \cos(\varphi_A) \cdot \cos(\varphi_B) \cdot \sin^2\left(\frac{\lambda_B - \lambda_A}{2}\right)}, \tag{3.5.1}$$

where φ_A and φ_B represent the latitudes, λ_A and λ_B represent the longitudes, r represents the radius of the sphere and d is the distance between the points. Features such as "consecutive

missing stations" and "time since last detection days" were also included, although they were less important in distinguishing anomalies. These features captured periods of unexpected gaps in detections or longer periods without detections, which could indicate possible problems with equipment or abnormal fish behaviour.

Table 3.3: Description of original and engineered features in the telemetry dataset comprising acoustic detections of dusky kob *Argyrosomus japonicus* tagged and monitored in the Breede Estuary, South Africa.

Feature description	
Original Features	
`fishid`	ID code of each individual animal.
`receiver`	The serial number of a specific receiver.
`station`	Name of the station.
`lat`	Geographic coordinate of the acoustic receiver that detected the fish.
`lon`	Geographic coordinate of the acoustic receiver that detected the fish.
`date`	The date on which an individual fish was detected.
`time_sa`	The time (UTC+2) at which an individual fish was detected.
Engineered Features	
`distance`	The distance between two points on the Earth's surface using the Haversine formula.
`duration in same station`	The time a fish remained at the same station before moving to another station or going undetected.
`num detections`	The total number of times a fish was detected during the study period.
`num days detected`	The number of separate days on which a fish was detected in the Breede Estuary.
`num unique stations`	The number of different stations on which a fish was detected throughout the Study period.
`consecutive missing stations`	A count of the number of consecutive stations that did not detect a fish, indicating periods when the fish moved out of detection range or through areas without receiver coverage.

The results demonstrate that engineered features, particularly "num unique stations" and "distance" played a pivotal role in enhancing the model's anomaly detection capabilities. These features enabled the NN-AE to identify subtle deviations in fish movement patterns, including complex, slowly evolving anomalies that would have otherwise been challenging to detect. This highlights the importance of feature engineering in optimizing the model's performance. Some of these engineered features such as number of detections and days detected, are commonly used in acoustic telemetry (e.g. Flávio et al. (2021);Cowley et al. (2008)).

3.6 Labelling Process

A detailed labelling process is first required to prepare the data for effective anomaly detection. The labelling process is based on three primary criteria, each capturing different patterns of abnormal fish movement.

- *If an individual fish was recorded at only one station throughout the study period, then it was flagged as an anomaly:* This criterion identifies anomalies where an individual fish was detected exclusively at a single station throughout the entire observation period. Such a pattern suggests either a malfunctioning tag or an uncharacteristic behaviour, such as the fish being stationary due to predation or illness. Figure 3.2 shows a horizontal line at station 2, indicating that the fish was only detected at this station without any movement to other stations over an extended period of time. Given prior knowledge of a species' movement behaviour, one can identify this as an unlikely movement. Figure 3.3 shows fish movement of fish A69-9001-23702 was only detected at one station, suggesting it remained stationary for the duration of its 2 day monitoring period, which is unlikely for this species (Griffiths, 1996).

Figure 3.2: The plot shows a fish that remained at only one station continuously over a prolonged period, indicating potential anomalies such as tag malfunction, illness, or unusual behaviour.

- *If an individual fish moved as per normal, but then remained at the same station for more than 120 days:* This criterion flags anomalies where a fish, after initially moving between stations as expected, remains at one station for more than 120 days. This prolonged stay could indicate issues like the fish being trapped or environmental factors preventing its movement. Figure 3.4 shows this scenario with a vertical drop to station 2, followed by a prolonged horizontal line, representing the extended stay at that single station.

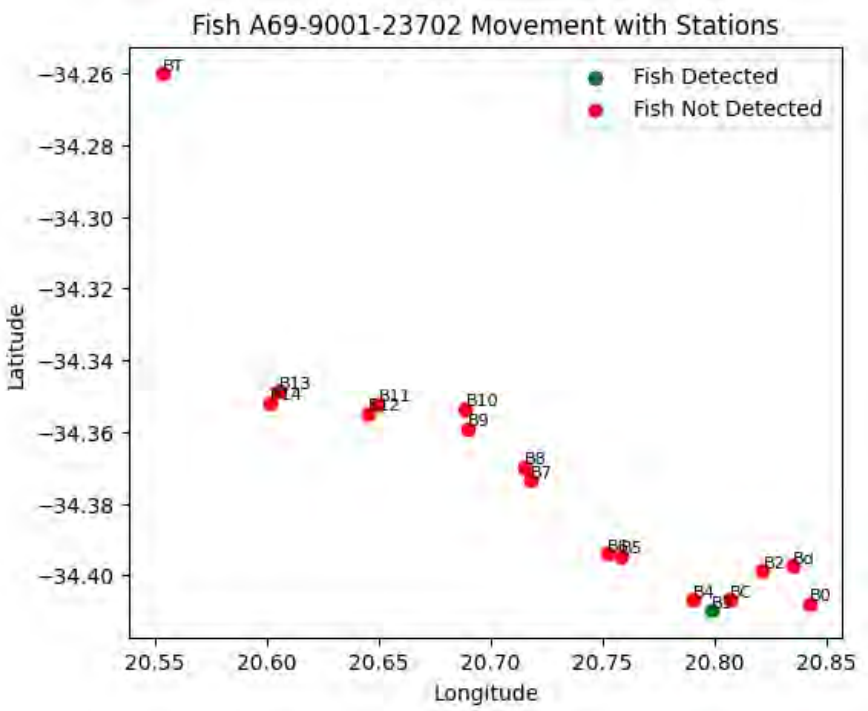


Figure 3.3: Patterns of detection for fish A69-9001- 23702 as an example of an anomalous detection pattern where a fish was only detected at one station, suggesting it remained stationary throughout the study period.

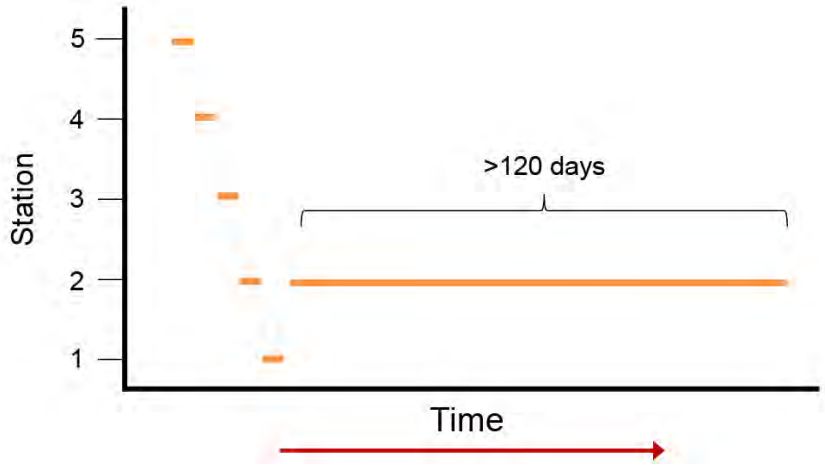


Figure 3.4: The plot shows a fish's movement through multiple stations before settling at station for over 120 days. Such extended residency at a single station is flagged as an anomaly.

- *If an individual fish was not detected by consecutive stations and the individual was missed by more than one consecutive station:* This criterion captures anomalies where a fish is not detected at consecutive stations (missed by more than one consecutive station) it should have passed through, which could imply tag transmission issues or unusual movement patterns. Figure 3.5 shows this criterion with the fish moving through stations but missing detections at several consecutive points, indicating irregularities in its movement or potential detection issues. Figure 3.6 depicts fish

movement of fish A69-9001-23769 showing an irregular movement pattern, detected at only three of 16 stations with a significant gap of seven consecutive missed stations, flagging it as an anomaly.

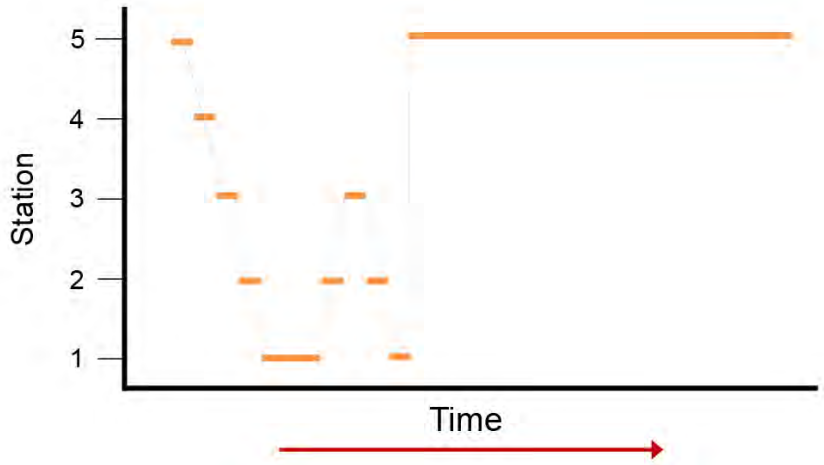


Figure 3.5: The plot shows a fish's movement through multiple consecutive stations before it jumps from station 1 to station 5 missing more than one consecutive stations. This type of movement is classified as an anomaly, and the station where the unusual transition occurs is marked as an anomalous event.

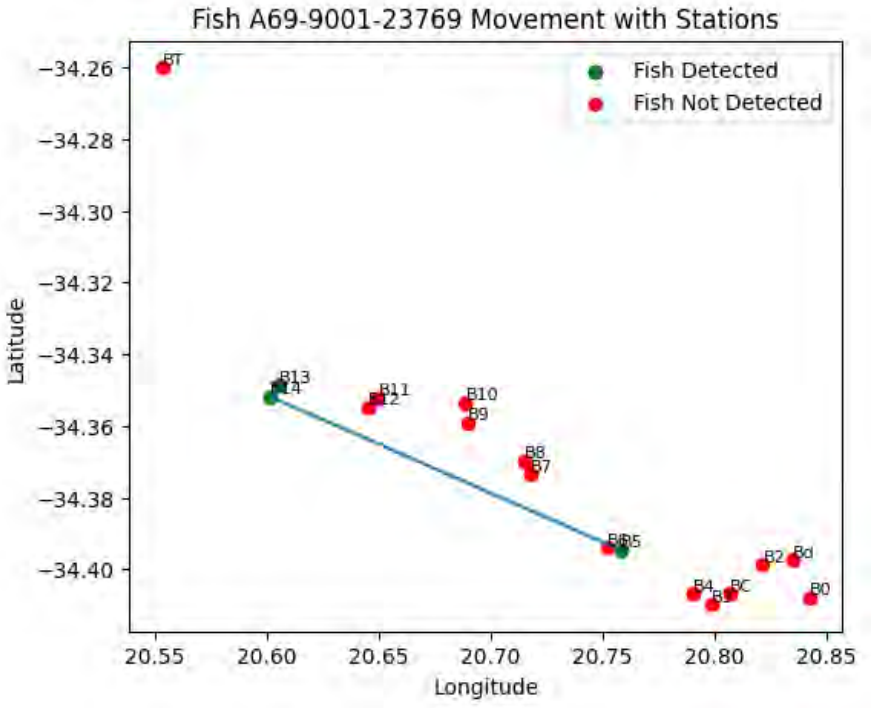


Figure 3.6: An example of an anomalous detection pattern, where fish A69-9001-23769 was only detected at three out of 16 stations in the estuary, but not in a consecutive manner. This pattern deviates from the defined notion of normality, as the fish was missed by multiple consecutive stations, flagging it as an anomaly.

Any detection that meets at least one of the defined criteria mentioned above is marked

as an anomaly (label 0), while those that do not meet any criteria are considered normal (label 1). To ensure data quality and a robust foundation for an anomaly detection system, the labelled data underwent validation, and detected anomalies were manually reviewed by a domain expert to confirm their validity. The complete flowchart of labelling is visually depicted in Figure 3.7.

The initial dataset contained 3038671 detections. During data pre-processing, duplicates were removed, resulting in 3013930 unique detections. After labelling, 2749960 detections (91.2%) were identified as "normal", while 263970 (8.8%) were classified as "anomalies". The significant imbalance of 91.2% normal to 8.8% anomalies led to the choice unsupervised learning classifiers, which do not require balanced datasets, unlike supervised classifiers. Unsupervised classifiers were trained to recognize only normal patterns. When tested with both normal and anomalous patterns, they fail to recognize the anomalies, allowing such detections to be flagged. Although unsupervised classifiers typically require less manual data preparation, in this study, the data were labelled to specifically train the models on normal patterns and then used the anomalous patterns to interpret the classifiers' results.

The anomalies were further analyzed by class. Of the 263970 anomalies detected, 259699 (98.4%) fell under the criterion *"If an individual fish was recorded at only one station throughout the study period, then it was flagged as an anomaly."* The remaining 4271 anomalies (1.6%) met the criterion *"If an individual fish was not detected by consecutive stations and was missed by more than one consecutive station."* No anomalies matched the criterion *"If an individual fish moved as per normal, but then remained at the same station for more than 120 days."* This analysis showed a highly skewed distribution, with one extreme case dominating the results, which may limit the model's ability to detect other types of anomalies. This limitation is discussed in more detail in Chapter 6.

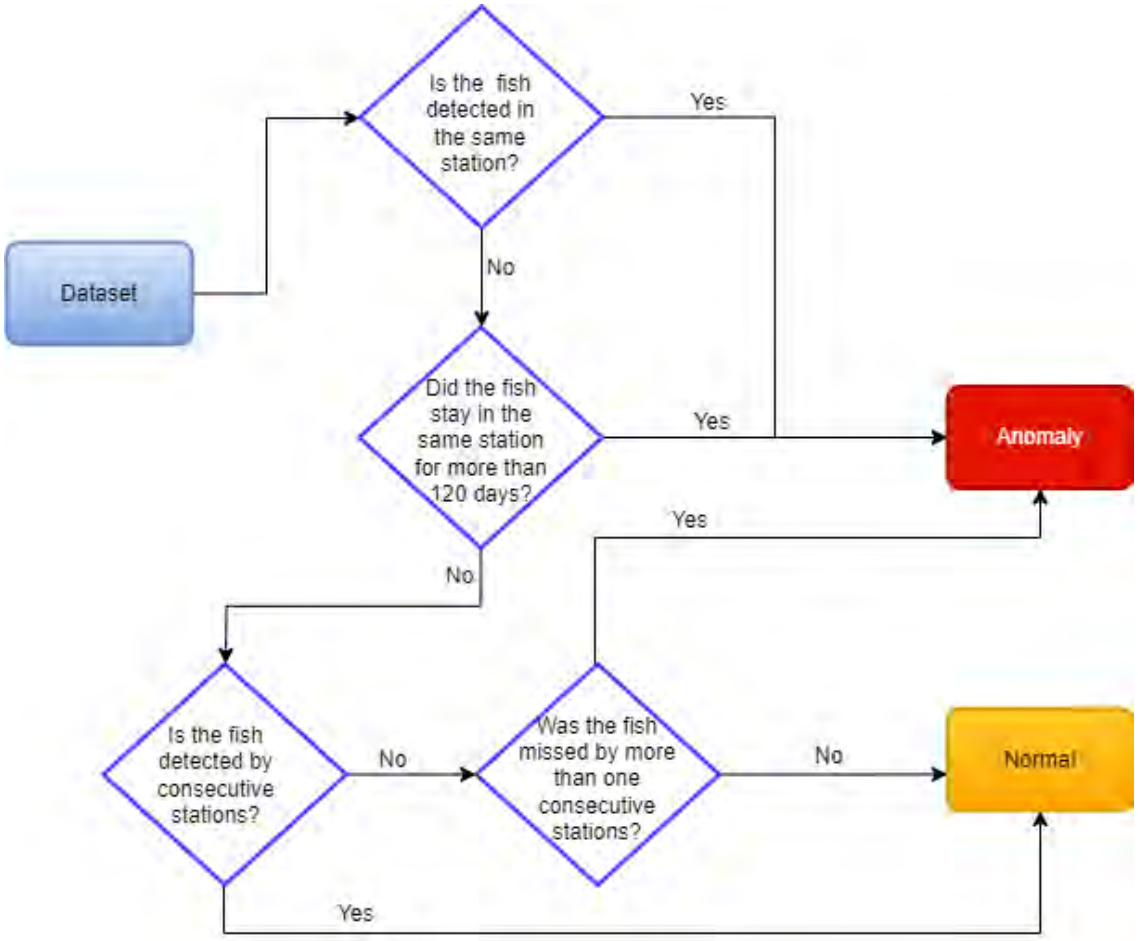


Figure 3.7: Flowchart for anomaly detection in acoustic telemetry data. This figure illustrates the detailed labelling process used to prepare the dataset for anomaly detection in acoustic telemetry data from 50 dusky kob tagged in the Breede Estuary between 2016 and 2021.

3.7 Resampling acoustic telemetry data from real data

The sampling rate is determined by the frequency of normal and anomalous detections at specific times. In this study, the acoustic telemetry data is sampled irregularly on a daily basis, and varies each day over the observation period. Figure 3.8 illustrates the detection of the same fish over three days. The horizontal axis represents the time of day when the fish was detected, clearly indicating that detections do not occur at regular intervals. These irregular detection intervals result from the combination of fish movement patterns and receiver detection probabilities. Although this is complete observational data (not missing values), inconsistent timing could lead to artifacts in time-series analysis if not properly adjusted. Since we train unsupervised classifiers only in normal patterns, we employ a resampling strategy that adjusts the irregular sampling of normal samples. This strategy aims to approximate the Nyquist sampling requirements, not necessarily to balance the data. A consistent sampling rate is vital for preserving the signal's integrity, crucial for precise analysis. Well-sampled data allows models to learn more effectively, providing a

reliable representation of the underlying signal, which in turn facilitates feature extraction and improves model performance.

The sampling strategy actively adjusted the sampling rate to reflect the daily changes in data collection. This adjustment was based on calculating the minimum of the shortest non-zero time intervals between consecutive data points, labelled as $\Delta^d t$, where the subscripts d indicates that this varied per day:

$$\Delta^d t = \min(\{t_{i+1} - t_i\}), \quad i = 1, \cdots, N_t, \tag{3.7.1}$$

where t_i and t_{i+1} represent the consecutive timestamps of data points during day d and N_t is the number of normal detection timestamps. To standardize the sampling rate across all days, we chose the minimum $\Delta^d t$ across days:

$$\Delta t = \min(\{\Delta^d t\}), \quad d = 1, \cdots, N, \tag{3.7.2}$$

where N is the total number of monitored days. The sampling rate is then the inverse of Δt given as:

$$f_s = \frac{1}{\Delta t}. \tag{3.7.3}$$

This sampling rate set the frequency at which data points were collected, helping to generate new data points at approximately consistent intervals.

Figure 3.8 shows that normal detections for the same fish are irregularly sampled on a daily basis. After implementing the sampling strategy described in Equation 3.7.3, we observed a pattern that approximates regular sampling. However, this was accompanied by computational challenges due to the high sampling rate, which required significant computational resources to train the classifiers. Instead of using the minimum rate as specified in Equation 3.7.3, I iteratively resampled using the next smallest rate until we found a trade-off between sampling regularity and computational feasibility for training the classifiers. With the adjustment following the trade-off search, the data in Figure 3.8 exhibited an approximate regularity as shown in Figure 3.9.

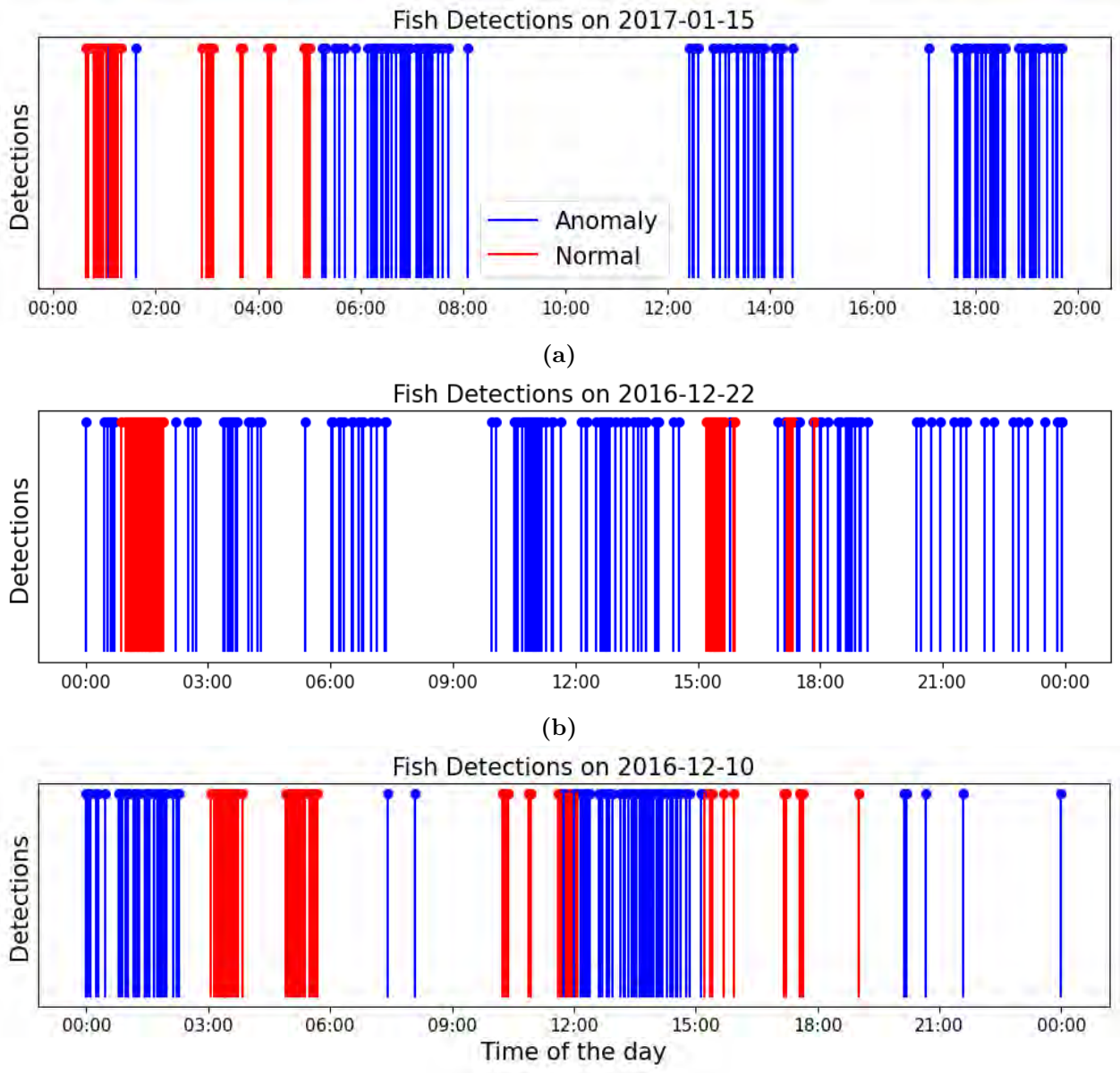


Figure 3.8: Examples of fish detections across three different days 2017-01-15 (a), (2016-12-22 (b) and 2016-12-10(c)) show irregular sampling patterns in acoustic telemetry data. The horizontal axis represents the time of day, while the vertical markers differentiate between normal (red) and anomalous (blue) detections. The varying frequency and irregular intervals of detections show the challenges of missing data and the importance of resampling strategies to preserve signal integrity and improve unsupervised classifier performance.

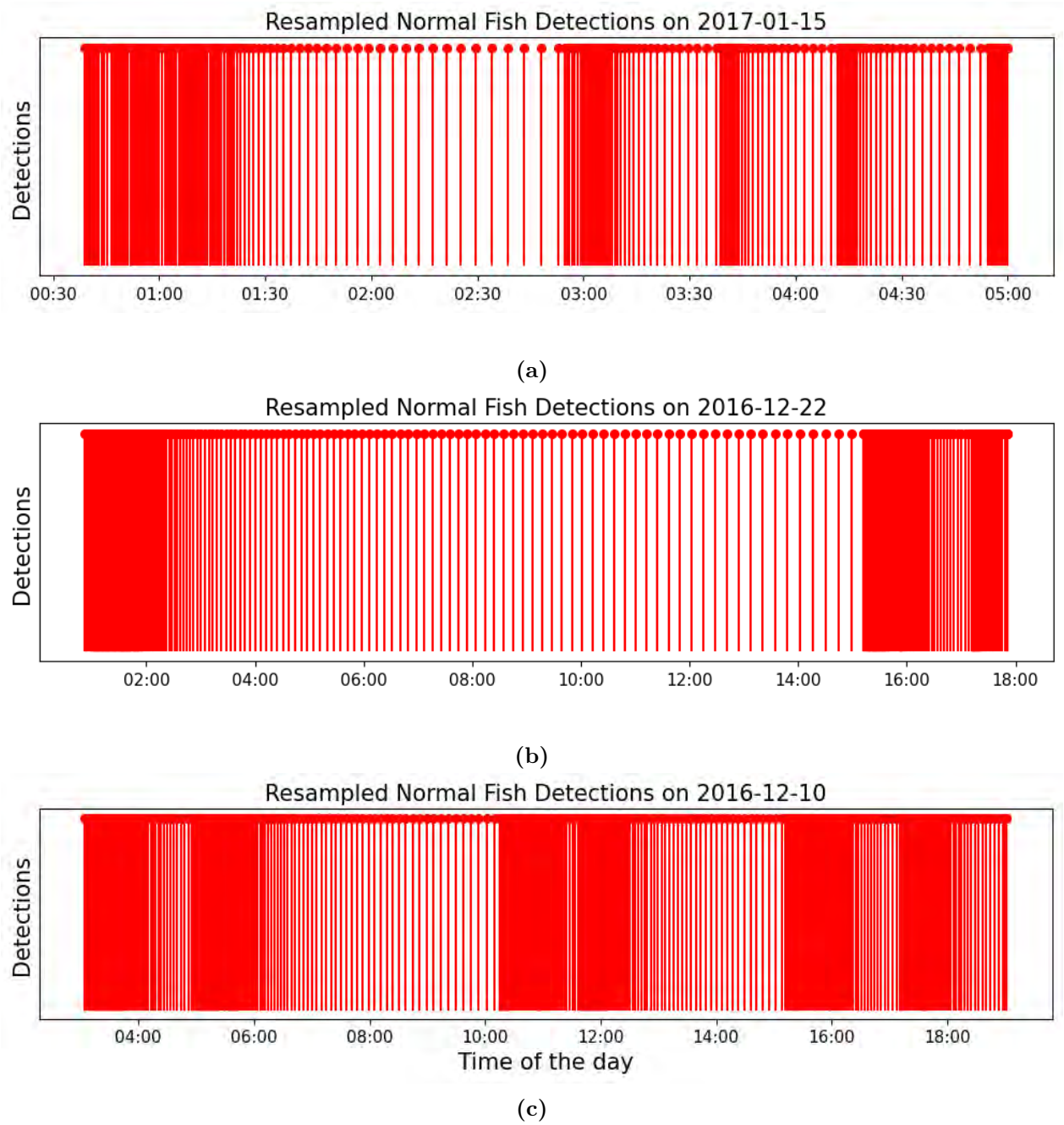


Figure 3.9: Comparison of resampled normal fish detections across three different days: 2017-01-15 (a), 2016-12-22 (b) and 2016-12-10 (c). Each day is resampled with the smallest sampling rate among the three days, approximating regular intervals. The resampling strategy is designed to adjust for irregular sampling patterns and ultimately improve the performance of unsupervised classifiers.

3.8 Summary

This chapter describes the methodology used to prepare and analyze the acoustic telemetry dataset. The data was labelled as normal or anomalous based on predefined criteria, with 91.2% of the data labelled as normal and 8.8% as anomalies. The labelling process involved identifying patterns in the data that deviated from normal behaviour, such as irregular

movement patterns or a prolonged stay at a single station. To address the irregular sampling rate, a resampling strategy was used to approximate a regular sample.

Chapter 4

Methodology

This chapter describes the methods used to detect anomalies in the telemetry dataset. Section 4.1 focuses on the design and implementation of the NN-AE, including the procedure for determining optimal thresholds and splitting the data into training, validation, and test subsets. The strategies for tuning the hyperparameters are explained in more detail in Section 4.1.3. Section 4.2 discusses the performance metrics used to evaluate the models.

4.1 Neural Network Autoencoders

In this section, we propose a creative approach for anomaly detection in time-series telemetry data by combining the strengths of NN techniques with the specialized architecture of AEs. The reason for choosing AE is their effectiveness in processing time-series data, which is characteristic of telemetry datasets.

Artificial neural networks are computational models inspired by the human brain, consisting of layers of interconnected nodes (neurons) that process and learn from data (Nwadiugwu, 2020). An autoencoder consists of an encoder, a latent space, and a decoder. The encoder compresses the input into a latent-space representation, and the decoder reconstructs the input data from this representation assuming a reconstruction error (Hinton & Salakhutdinov, 2006; Chen et al., 2018; Sakurada & Yairi, 2014). The reconstruction error, or the difference between the original and reconstructed data, serves as the primary indicator of anomalies (Chen et al., 2018). One of the challenging tasks for AEs is determining the optimal threshold to bound the reconstruction errors. In this work introduced an additional block that finds this optimal threshold for the reconstruction error, and any data point exceeding this threshold was considered anomalous.

This approach used the strengths of NN-AE to detect anomalies in a telemetry dataset (Figure 4.1). AEs are particularly well suited for learning a compressed representation of complex data, which enables effective anomaly detection. By using a NN-AE, the model captured

complex patterns and relationships within the data, allowing for accurate identification of anomalies. This approach provided a powerful tool for dealing with the complexity of telemetry datasets. The proposed NN-AE model is a powerful anomaly detection tool that

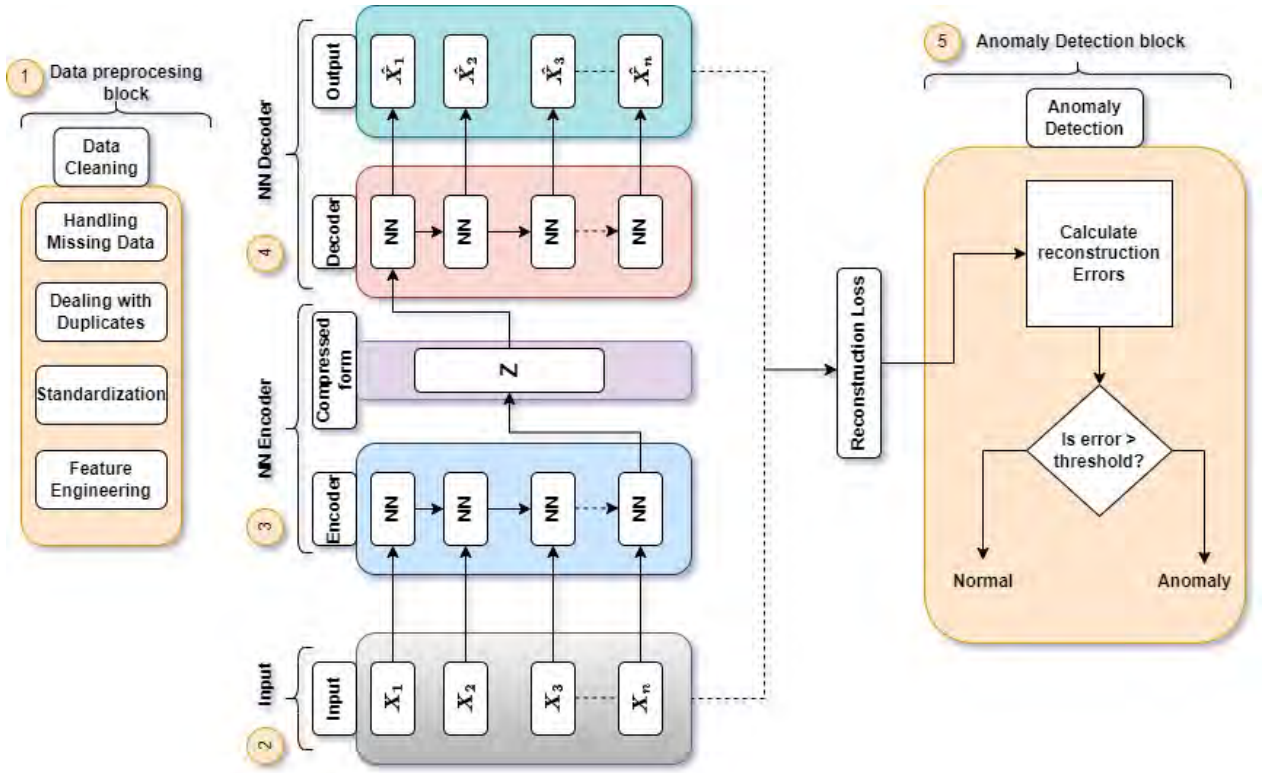


Figure 4.1: The diagram shows a NN-AE for anomaly detection. It includes data pre-processing, encoding to a compressed form, decoding, and an anomaly detection block that flags anomalies based on reconstruction errors.

consists of an encoder and a decoder. The encoder takes a sequence of observations as the input and compresses them into a compact latent space representation that captures the important features and patterns in the data. This compressed representation is then fed into the decoder, which reconstructs the original input data using the learned patterns and relationships. The difference between the original and reconstructed data is known as the reconstruction error that serves as a key indicator for identifying anomalies (Chen et al., 2018). Both the encoder and the decoder consist of several fully connected layers with decreasing and increasing dimensionality, respectively, which enables efficient information processing and reconstruction.

The proposed NN-AE model also included an anomaly detection block to identify deviations from the normal data pattern. An anomaly, in this instance, was defined as an observation that deviated significantly from what was considered to be normal behaviour. Within this block, we calculated the reconstruction error for each data point, which indicated how much the reconstructed input deviated from the original input. An optimal threshold was then set, which served as a decision boundary and allowed us to distinguish between normal and abnormal behaviour in the data (Chen et al., 2018). By setting a threshold, we established

a criterion for what was considered normal behaviour. Any data points that exceeded this threshold were flagged as anomalies. The process of selecting the optimal threshold is discussed in Subsection 4.1.1.

4.1.1 Finding the optimal threshold

Among the metrics of precision, recall, and specificity we aim to maximize the recall, as this indicates that the model has maximized the detection of true anomalies while minimising the prediction of anomalies as normal instances (i.e., minimising FN). This is crucial for telemetry observations, as the presence of FN in the data might skew the scientific interpretation for which the telemetry data were observed. The goal is to achieve a recall of 1, which ensures that no FN are present in the observations. For the range of thresholds where recall is maximized, we then select the threshold that also maximizes both precision and specificity to reduce the identification of false anomalies (i.e. minimising FA).

The algorithm works as follows. Assume that the percentiles run from $i = 1, \ldots, n$ where each percentile corresponds to a specific threshold. Let $R = \{r_i\}$ be the recall set where each r_i is the recall for percentile i, $P = \{p_j\}$ be the precision set where j corresponds to selected indices from R, and $S = \{s_k\}$ be the specificity set where k corresponds to selected indices from P. The algorithm consists of three steps: the first step finds the maximum values in R; $R_{\max} = \max\{r_i : i = 1, \ldots, n\}$ and identify all indices j where $R_j = R_{\max}$, i.e. the indices of the maximum values in R; $J = \{j : R_j = R_{\max}\}$. The second step finds the maximum value in P for $j \in J$; $P_{\max} = \max\{p_j : j \in J\}$ and identifies all indices k where $P_k = P_{\max}$, i.e. the indices of the maximum values in P; $K = \{k : P_k = P_{\max} \text{ and } k \in J\}$. The final step finds the maximum value in S for $k \in K$; $S_{\max} = \max\{s_k : k \in K\}$ and produces all the indices l where $S_l = S_{\max}$. The best percentile is any of the l percentiles (see Algorithm 1).

Tables A.1, A.2, and A.3 (see Appendix A) show the optimal thresholds for NN-AE trained and validated on three datasets: the original dataset without resampling, with 90s resampling, and with 65s resampling, respectively. The optimal percentile thresholds were 65 for the model trained on the dataset without resampling, 67 for the 90s resampling dataset, and 69 for the 65s resampling dataset. This trend suggests that datasets with higher temporal resolution (i.e., shorter resampling intervals) required slightly higher percentile thresholds for optimal anomaly detection. Figure 4.2 shows the corresponding threshold for 65^{th} percentile, where the reconstruction errors are plotted against the indices of the validation dataset without resampling.

Algorithm 1 Finding the optimal threshold

Require: Model M
Ensure: Optimal threshold
1: Initialize arrays: $R \leftarrow [\,]$, $P \leftarrow [\,]$, $S \leftarrow [\,]$, `Percentiles` $\leftarrow [1, 2, \ldots, 100]$
2: **for** each percentile i in `Percentiles` **do**
3: $threshold \leftarrow i$-th percentile of reconstruction errors
4: Calculate evaluation metrics using model M:
- $R[i] = \frac{TP}{TP+FN}$
- $P[i] = \frac{TP}{TP+FP}$
- $S[i] = \frac{TN}{TN+FP}$

5: **end for**
6: **Step 1: Maximize Recall**
7: $R_{\max} = \max(R)$
8: $J =$ indices where $R == R_{\max}$ ▷ All percentiles with max recall
9: **Step 2: Maximize Precision** (from candidates in J)
10: $P_{\max} = \max(P[J])$
11: $K =$ indices in J where $P == P_{\max}$ ▷ Subset with max precision
12: **Step 3: Maximize Specificity** (from candidates in K)
13: $S_{\max} = \max(S[K])$
14: $L =$ indices in K where $S == S_{\max}$ ▷ Final optimal percentiles
15: **return** any percentile in L as optimal (e.g., median(L))

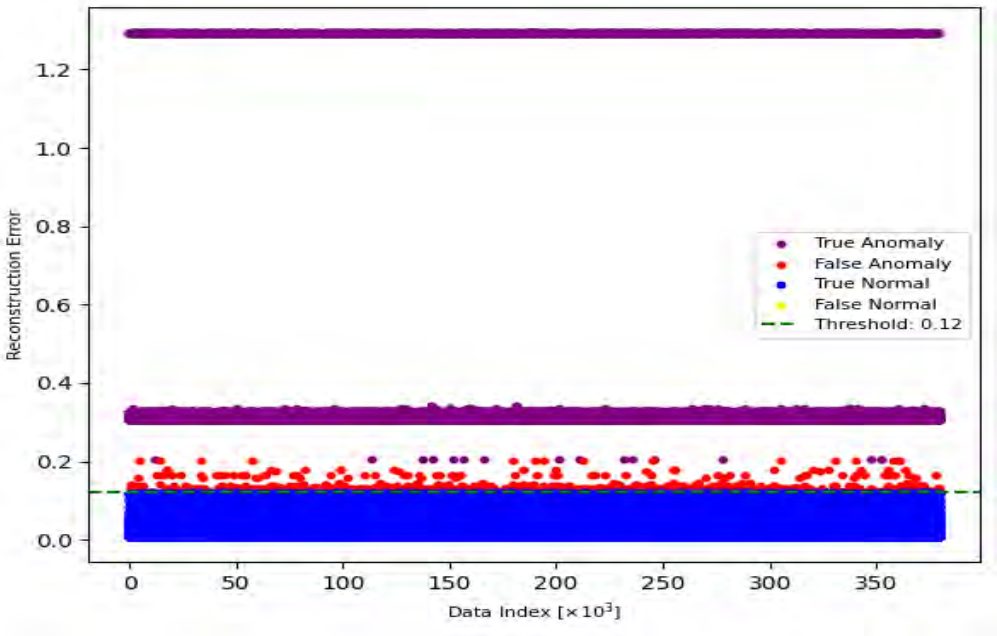


Figure 4.2: Reconstruction errors plotted against the data index, highlighting TN (blue), TA (purple), FA (red), and FN (yellow) using the validation datasets without resampling. Higher reconstruction errors indicates potential anomalies.

4.1.2 Splitting, Training, Validation and Testing

Training, validation, and testing are critical in the development of NN-AE because they ensure that the model accurately identifies anomalies and generalizes well to new, unseen data. To ensure the integrity of the time series, the chronological order of the fish detections was maintained in all splits. The validation set is used during the model development process to fine-tune the hyperparameters (Maleki et al., 2020). Figure 4.3 provides a comprehensive overview of the experimental design, encompassing each stage from data pre-processing and dataset splitting to resampling, training, hyperparameter tuning, and testing.

In *Step 1*, the data was pre-processed, including handling missing values, removing duplicates, and performing feature engineering. After pre-processing, the data moved to *Step 2* for various splits. Initially, the dataset was divided into two main categories: normal and anomalous samples. Normal samples accounted for 91.2% of the data (2749960 samples), while anomalous samples made up 8.8% (263970 samples) of the original dataset.

For the normal samples, 10% (274996 samples) was set aside for testing, and the remaining 90% (2474964 samples) underwent resampling as described in Section 3.7. The resampled data was then used to train the traditional models without further spitting. For the NN-AE model, the resampled data were further divided into 80% for training and 20% for validation. Note that the 10% test data were not resampled. This allowed the model's performance to be evaluated using realistic telemetry data, which by nature has irregular sampling. While resampling improved the stability of the training by regulating the temporal intervals, testing with the original data ensured the robustness of the model under real-world conditions. This approach is consistent with the ecological goal of detecting anomalies in unprocessed datasets where irregular sampling is unavoidable.

For the anomalous samples, 50% (131985 samples) was allocated for testing, while the other 50% (131985 samples) was used for training the traditional models and also served for validation to determine the autoencoder threshold. The splitting strategy outlined indicated that the traditional models were trained using both normal and anomalous data, while the NN-AE model was trained exclusively on normal data. The NN-AE was trained on normal data because autoencoders learn to reconstruct normal patterns, making it possible to detect anomalies based on reconstruction errors. In contrast, traditional models (IF, LOF, DBSCAN) are designed to be trained and evaluated on datasets containing both normal and anomalous instances, enabling them to generalize and identify outliers based on global or local data density without requiring a separate set of normal data. Training them only on normal data would reduce their effectiveness in real-world scenarios where anomalies are rare or not clearly labelled. Although NN-AE relies on unsupervised representation learning, traditional models detect anomalies using the inherent structure of the entire data set.

After *Step 2*, the data proceeded to *Step 3* for training and hyperparameter tuning. For each model, hyperparameter tuning was performed using the GridSearch algorithm Zahedi et al. (2021). The IF and LOF models were optimized for anomaly detection based on contamination and density metrics. DBSCAN was optimized for clustering, while NN-AE used a validation set to refine its configuration and threshold for anomalies. Details about the hyperparameters are provided in Section 4.1.3

After training and hyperparameter tuning, the models proceeded to *Step 4* for testing. The test data, set aside during the splitting step (prior to resampling and training), contained both normal and anomalous instances and was used to evaluate the models' ability to accurately detect anomalies.

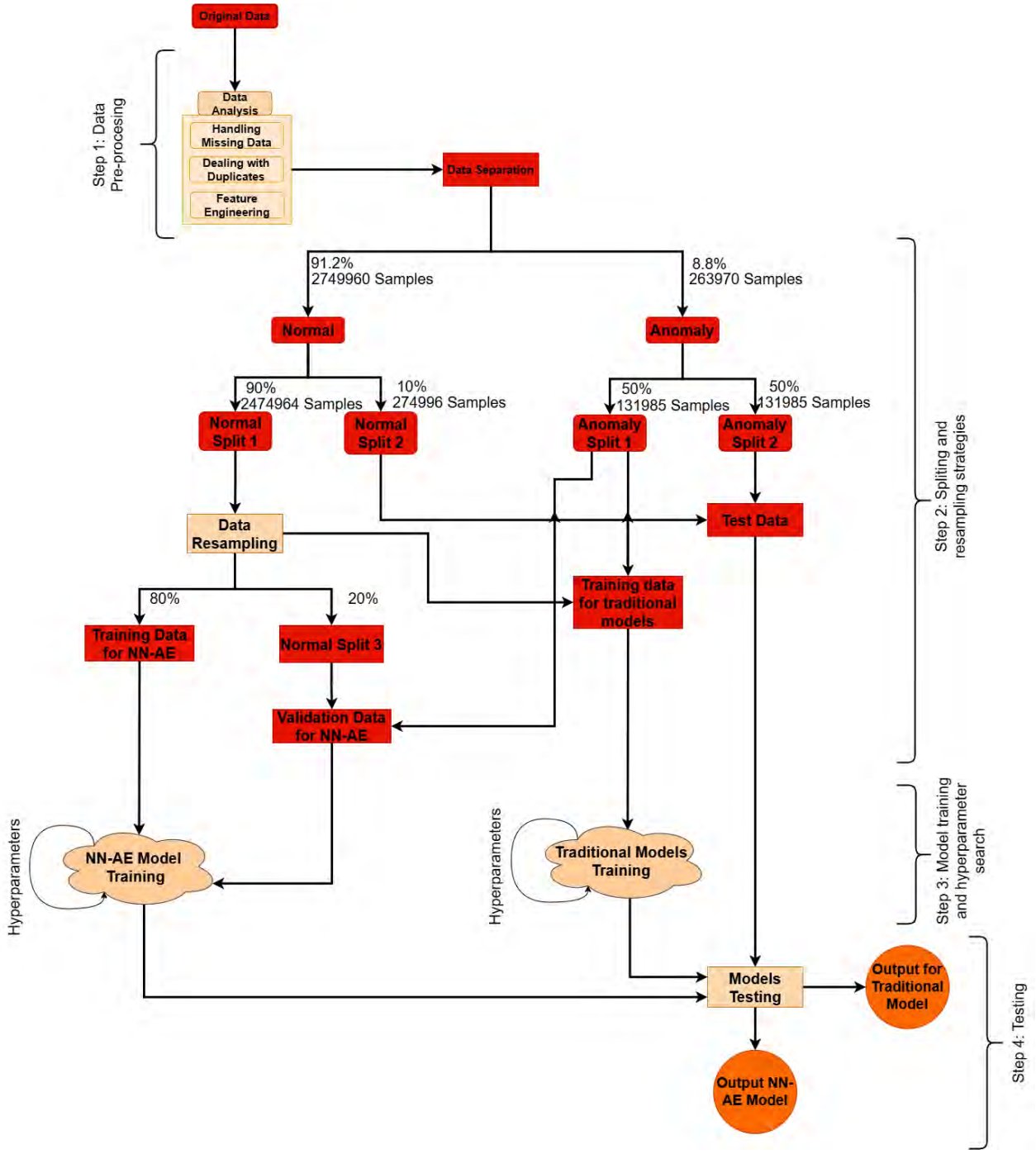


Figure 4.3: The diagram shows an anomaly detection workflow with data pre-processing, followed by data splitting into training and testing sets, and concludes with model training and evaluation for both the NN-AE and traditional models used in this work.

4.1.3 Hyperparameters

Hyperparameter tuning is an important part of training any ML model and is referred to as the task of selecting a set of optimal hyperparameters for a learning algorithm. The simplest approach to tuning hyperparameters is undoubtedly grid search. In this method, we simply constructed a model for each possible combination of the given hyperparameter values, evaluated each model and selected the model that gave the best results (Zahedi et al.,

Table 4.1: This table presents optimal hyperparameters selected for each anomaly detection model used in the work: IF, LOF, DBSCAN, and NN-AE. The models were tuned using various hyperparameter options, as listed in the "Hyperparameter" column. The "Best Parameter" columns show the selected hyperparameter values under three conditions: no resampling, resampling at 90s, and resampling at 65s.

Model	**Hyperparameter**	**Best Parameter**		
		No	**90s**	**65s**
IF	Contamination: [0.001, 0.02, 0.05, 0.3]	0.001	0.001	0.001
	Number of Estimators: [100, 150, 200]	100	100	100
LOF	Number of Neighbors: [5, 10, 20]	5	5	5
	Contamination: [0.01, 0.08, 0.1, 0.2]]	0.01	0.01	0.01
DBSCAN	Epsilon: [0.5, 1.5, 2, 3.5, 5]	0.5	0.5	0.5
	Minimum Samples: [2, 4, 10]	10	10	10
NN-AE	Learning Rate: [0.001, 0.01, 0.1]	0.001	0.001	0.001
	Units Layer: [4, 8, 16, 32, 64, 128]	128	128	128
	Compression Layer Units: [2, 4, 8]	2	2	2
	Batch Size: [128, 256, 512]	512	512	512
	Epochs: [20, 50]	50	50	50

2021).

For each model, hyperparameter tuning was performed and the IF model optimally detected anomalies by recursively partitioning data points with 100 estimators and a contamination setting of 0.001, balancing robustness and computational efficiency. The LOF identified anomalies by evaluating the local density deviation of each point relative to its neighbours, using 5 neighbours and a contamination parameter of 0.01. For DBSCAN, the optimal parameters were determined to be an epsilon value of 0.5 and a minimum sample size of 10. For the NN-AE, hyperparameter tuning was conducted using the validation set, which contained both normal and anomalous samples, in order to determine the optimal threshold for the autoencoder. The optimal configuration for the NN-AE consisted of a learning rate of 0.001, 128 units per layer, and 2 units in the compression layer. The model was trained with a batch size of 512 over 50 epochs, with an optimal threshold of 0.1207. Table 4.1 summarises the range of hyperparameters and their optimal values determined via GridSearch.

4.2 Performance metrics

We evaluated the models using several performance metrics, including the confusion matrix, Receiver Operating Characteristic (ROC) curve, accuracy, precision, recall, specificity, and f_1-score. The confusion matrix summarised the model's predictions into four categories: True Anomalies (TA) refers to the number of anomalous samples that were correctly identified as anomalous. False Anomalies (FA) are the number of normal samples incorrectly classified as

Table 4.2: Summary of key classification metrics used to evaluate model performance. Each metric provides distinct insights into the model's ability to correctly classify normal and anomalous instances.

Name	Description	Formula
Accuracy	The proportion of correctly classified instances among the total instances.	$\frac{TA+TN}{TA+TN+FA+FN}$
Precision	The ratio of true anomaly predictions to the total number of positive predictions, indicating how often the model is correct when it predicts a positive outcome.	$\frac{TA}{TA+FA}$
Recall	The ratio of true anomalies to the total actual anomalies, measuring the model's ability to identify positive instances.	$\frac{TA}{TA+FN}$
Specificity	The ratio of true normals to the total actual negatives, indicating how well the model identifies negative instances.	$\frac{TN}{TN+FA}$
F1 Score	The harmonic mean of precision and recall. It balances the two metrics, especially in cases of imbalanced class distributions.	$\frac{2\times\text{Precision}\times\text{Recall}}{\text{Precision}+\text{Recall}}$

anomalous. True Normals (TN) represents the number of normal samples that were correctly classified as normal, while False Normals (FN) were the number of anomalous samples that were incorrectly classified as normal. These values were the basis for calculating the other metrics in Table 4.2. The ROC curve is a graphical representation of a model's ability to distinguish between classes. It plots the true anomaly rate against the false anomaly rate at various threshold settings. The Area Under the Curve (AUC) indicated the model's overall performance, with a value of 1 representing a perfect model.

4.3 Summary

This chapter outlined the methodology for detecting anomalies in telemetry data using ML techniques. A NN-AE was trained to recognise normal movement patterns of the dusky kob and identify deviations indicative of anomalies. The NN-AE model was trained with normal data only, while traditional models (IF, LOF, DBSCAN) were trained with both normal and anomalous data. Hyperparameter tuning was performed using GridSearch to optimize performance. Threshold optimization was also performed to balance high recall and high specificity to ensure that FN and FA were minimised. It was found that the 65th percentile (threshold of 0.1207) performed best and provided a reliable threshold for anomaly

detection. The evaluation process included metrics such as confusion matrix, precision, recall, F1 score and ROC curve, which provided a comprehensive assessment of the model's ability to discriminate between normal and anomalous data.

Chapter 5

Results and Discussion

In this section, this study presents the performance results of various traditional ML models alongside the proposed model, all assessed using the same telemetry dataset.

5.1 Training

I trained three widely used traditional unsupervised classifiers for anomaly detection: IF, LOF, and DBSCAN. Additionally, we trained two DL based autoencoder models, specifically the NN-AE and LSTM-AE. Although the traditional models allowed quick training without extensive hyperparameter tuning, they were prone to high false normal rates (that is, they predicted anomalous as normal). This occurred because traditional models struggled to distinguish subtle deviations in anomalous patterns, and because some of the normal behaviour shared features with anomalous patterns. Such as fish that spent 100 to 119 days at a single station (which is considered normal behaviour) had values in the "duration in same station" feature that were close to the anomaly threshold of 120 days. As a result, a traditional model incorrectly classify these cases as anomalies. However, traditional models demonstrated competitive performance compared to NN-AE in minimising false anomalies (i.e., predicting normal instances as anomalous). The LSTM-AE quickly overfitted the data, even after resampling efforts. Due to the limitations observed with the LSTM-AE model, this paper focuses exclusively on the performance of traditional models and NN-AE as baseline models for the detection of telemetry anomalies.

5.1.1 Performance

Using the optimal parameters, the confusion matrix generated from the classifiers' predictions on the test set is shown in Figure 5.1, with the top panel depicting models trained on data without resampling, and middle and bottom panels showing models trained on resampled data at 90s and 65s, respectively. Figure 5.2 provides a detailed visualization of TA, FA, TN, and

FN derived from the confusion matrix of the data without resampling, with 90s resampling, and with 65s resampling. In addition, Figure 5.3 provides a detailed visualization of the key performance metrics (precision, recall, accuracy, f_1-score, and AUC) for different models (NN-AE, IF, DBSCAN, LOF) evaluated on data without resampling, with 90s resampling, and with 65s resampling. These results offer a comprehensive overview of the models' performance, warranting a thorough discussion to evaluate their strengths and weaknesses in classifying normal and anomalous detections.

The results revealed that IF outperformed other models by eliminating FA, with NN-AE minimizing FA compared to LOF and DBSCAN. Resampling at finer rates decreased FA for NN-AE but not for other models, reflecting NN-AE's enhanced performance through increased data size and high-resolution data. This improvement stems from the model's ability to capture subtle patterns, reduce overfitting, and generalize more effectively with increased data size, unlike LOF and DBSCAN, which reach performance saturation quickly where additional data no longer yields significant improvement ramlakhan (2024). In telemetry studies, these findings are noteworthy. As telemetry datasets grow continuously with each new recording, a model capable of handling large datasets is essential, and minimizing FA is important. Detecting a high rate of FA would mean that a substantial portion of normal data is misclassified as anomalous, and therefore removed from the telemetry observation. The removal of a substantial portion of normal data could skew the primary analyses for which the telemetry data are collected for.

The results indicate that NN-AE achieves a 100% TA detection rate (i.e. no FN), which is crucial for telemetry observations, as it ensures that all anomalies are eliminated, leaving only normal samples. However, a very small fraction of FA; i.e. ~ 0.00352 of normal samples was also misclassified as anomalies and subsequently removed. This minimal removal of normal samples had no significant impact on data quality, as telemetry observations operate at a big data scale, making the loss of such a small fraction of data negligible.

Figure 5.4 presents the performance metrics of the models, including accuracy, F_1-score, recall, precision, specificity, and AUC, with error bars indicating the confidence intervals for each metric. A narrow confidence interval suggests a stable estimate, meaning repeated data reshuffling and splitting would yield similar results. In contrast, a wider confidence interval implies an unstable estimate, reducing its reliability. The results clearly demonstrate that NN-AE outperformed the other models with only specificity from other models competing comparably. This was attributed to all models showing a capability to minimize the FA, as discussed in Section 4.1.1. The values used to generate Figure 5.4 are detailed in Table 5.1, which provides a comprehensive breakdown of each metric and includes information on the computational requirements and confidence intervals of each model.

Training times generally increase under resampling conditions. For the IF model, training times range between 4.17 minutes and 4.74 minutes. For LOF models, training times range

from 24.61 minutes to 32.48 minutes. The DBSCAN model requires training times between 8.92 minutes and 9.44 minutes. All NN-AEs require significant training time, ranging from 11.6 hours to 14.2 hours. While these computation times may seem high during training, NN-AEs are generally fast when deployed. All experiments were conducted on a system with a 12th Gen Intel® Core™ i7-12700 CPU @ 2.10 GHz, 16 GB RAM, running Windows 11 Enterprise (Version 23H2) with Python 3.12.7, TensorFlow 2.18.0, and scikit-learn 1.5.1.

Table 5.1: Performance comparison of classifiers without resampling and with resampling at 90s and 65s intervals. Metrics evaluated include accuracy, recall, specificity, precision, F_1-score, AUC, and the number of parameters and training time.

Classifiers	**IF**			**DBSCAN**			**LOF**			**NN-AE**		
Resampling	**NO**	**YES**		**NO**	**YES**		**NO**	**YES**		**NO**	**YES**	
		90s	**65s**		**90s**	**65s**		**90s**	**65s**		**90s**	**65s**
Accuracy	67.60±0.0011	67.61± 0.0012	67.62±0.0012	66.00±0.0011	66.00±0.0012	66.00±0.0012	66.83±0.0013	66.83±0.0012	66.83±0.0013	99.76±0.0013	99.77±0.0002	99.85±0.0002
Recall	0.10±0.0001	0.15± 0.0002	0.16± 0.0002	0.24±0.0002	0.24±0.0002	0.24±0.0002	0.40±0.0003	0.40±0.0003	0.40±0.0003	100±0.0000	100±0.0000	100±0.0000
Specificity	100±0.0000	100±0.0000	100±0.0000	97.57±0.0002	97.57±0.0005	97.57±0.0004	98.71±0.0004	98.71±0.0003	98.71±0.0003	99.64±0.0002	99.66±0.0003	99.78±0.0003
Precision	100±0.0000	100±0.0000	100±0.0000	4.69±0.0036	4.69±0.0040	4.69±0.0040	13.16±0.0092	13.16±0.0091	13.16±0.0084	99.27±0.0092	99.31±0.0005	99.55±0.0006
F_1-score	0.21±0.0003	0.30±0.0003	0.33±0.0004	0.47±0.0004	0.47±0.0004	0.47±0.0005	0.78±0.0006	0.78±0.0005	0.78±0.0006	99.63±0.0005	99.65±0.0002	99.77±0.0003
AUC	50.05±0.0001	50.07±0.0001	50.08±0.0001	48.90±0.0003	48.90±0.0003	48.90±0.0002	49.56±0.0003	49.56±0.0002	49.56±0.0002	99.82±0.0001	99.83±0.0001	99.89±0.0001
No. of parameters	5	5	5	2	2	2	5	5	5	3597	3597	3597
Train time (minutes)	4.17	4.55	4.74	8.92	9.13	9.44	24.61	26.73	32.48	699.65	737.97	852.87

A temporal analysis of FA, TA, FN, and TN provided additional insights into model performance, with results visualised in subsection 5.1.2. The subsection below highlights fish movement patterns between 2016 to 2021, providing a detailed temporal perspective on the behaviour and detection capabilities of the models across various time intervals.

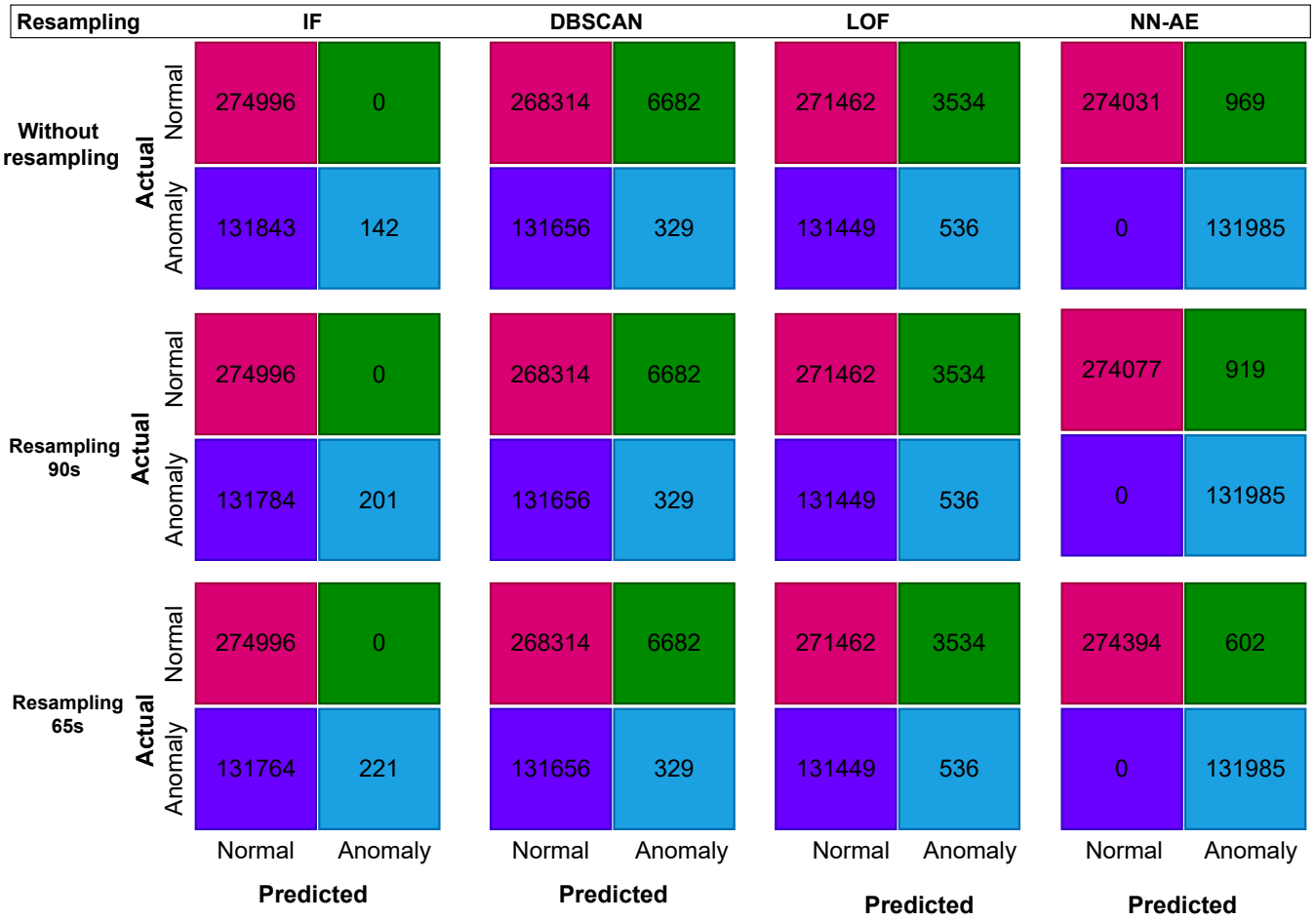


Figure 5.1: Confusion matrices for different anomaly detection models (NN-AE, DBSCAN, IF, LOF) applied to telemetry data, comparing "Normal" and "Anomaly" classifications at different sampling rates (65s and 90s).

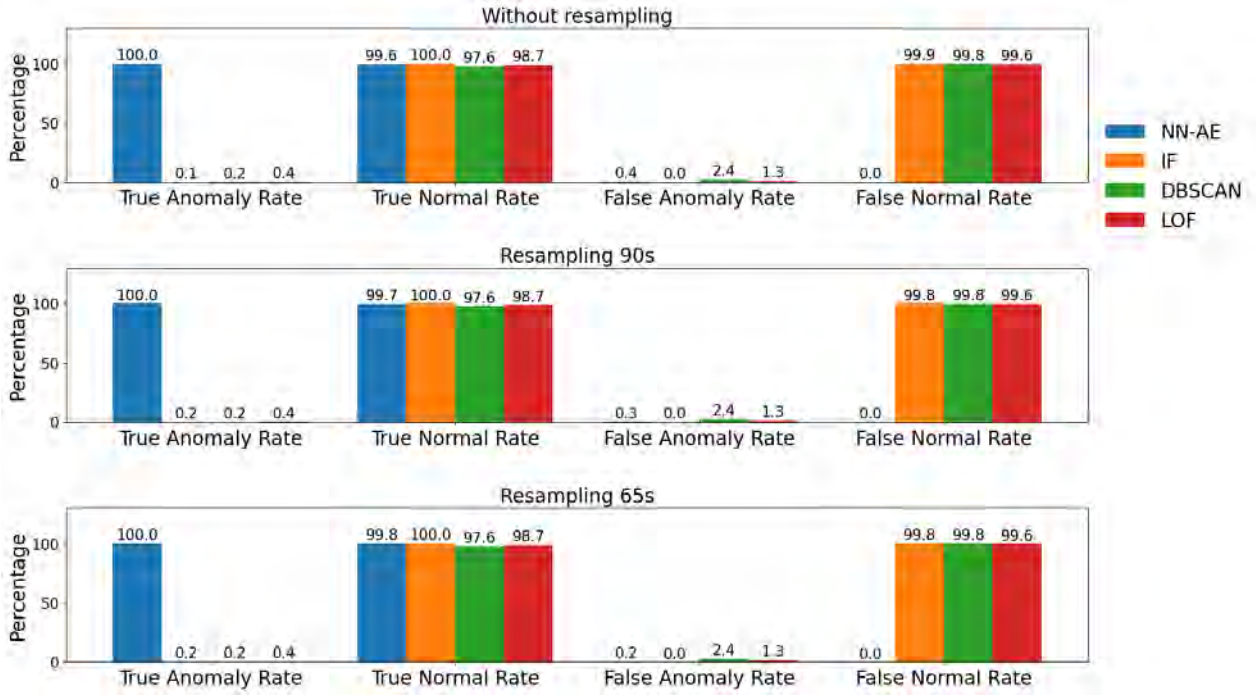


Figure 5.2: This bar chart shows the key metrics of the confusion matrix TA, TN, FA, and FN for anomaly detection models (NN-AE, IF, DBSCAN, LOF). The performance is compared across three conditions: without resampling, with 90s resampling, and with 65s resampling. NN-AE achieves the highest TA and TN rates, indicating its strong performance in anomaly detection. IF has a higher false normal rate, while DBSCAN and LOF have balanced but slightly lower detection capabilities.

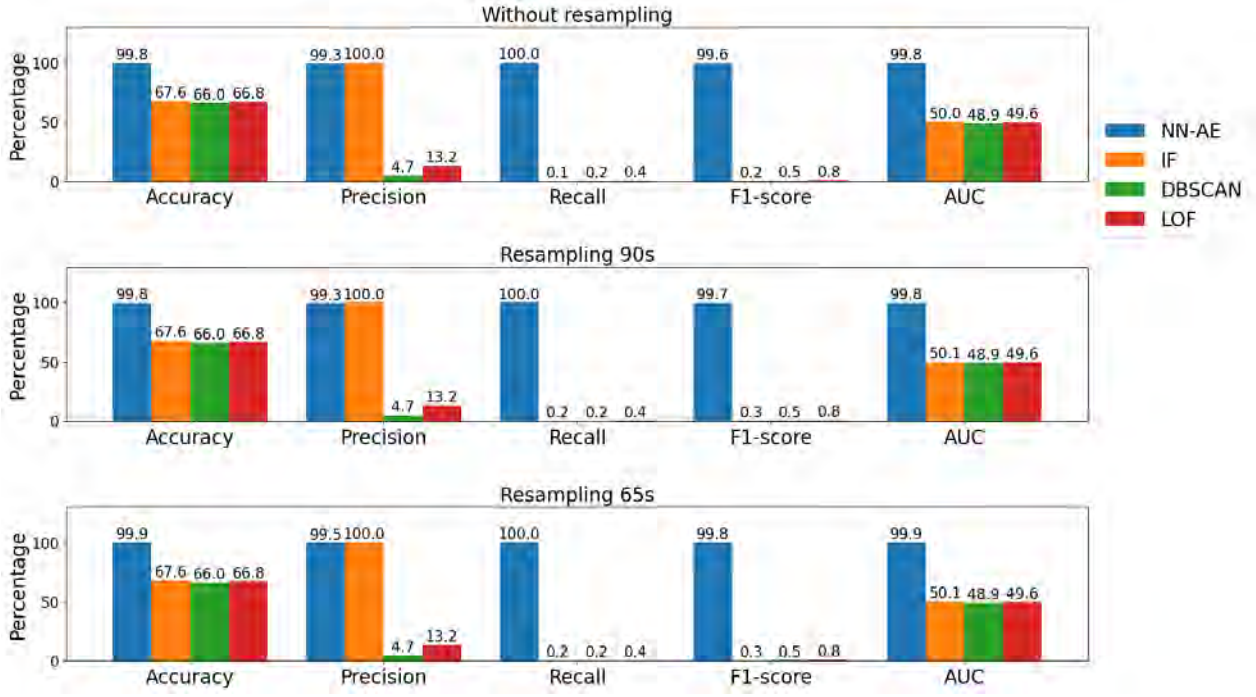


Figure 5.3: This bar chart shows the key performance metrics (precision, recall, accuracy, f_1-score, and AUC) for different models (NN-AE, IF, DBSCAN, LOF) evaluated on data without resampling, with 90s resampling, and with 65s resampling. The NN-AE model consistently achieves near-perfect performance, significantly outperforming the traditional models.

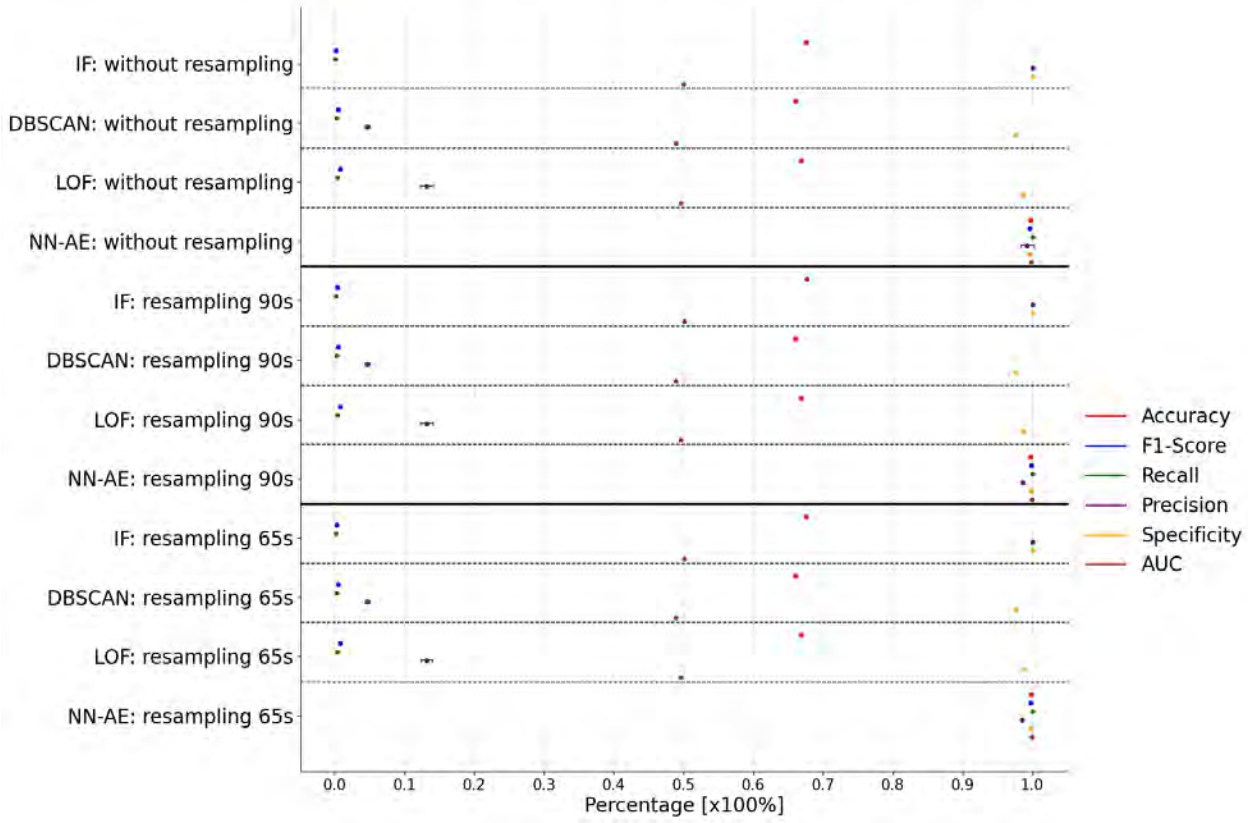


Figure 5.4: This figure compares the performance of different anomaly detection models (LOF, DBSCAN, IF, and NN-AE) across six metrics: accuracy, f_1-score, recall, precision, specificity, and AUC, along with confidence intervals. Model variations (e.g., LOF_65s, DBSCAN_90s) show differing levels of performance on these metrics across varying sampling rates. NN-AE performs consistently well in AUC and accuracy, while the other models exhibit more variation in Precision and Recall.

5.1.2 Temporal analysis of movements and detections

Figures 5.5, 5.6, 5.7, and 5.8 illustrate the movement patterns of adult dusky kob in the Breede Estuary, highlighting anomalies and TA as detected by the NN-AE, IF, LOF, and DBSCAN algorithms. Each figure consists of two parts: (a) provides an overview of dusky kob movements from 2016 to 2021, represented by blue lines indicating latitudinal shifts over time. Red dots denote anomalies and magenta crosses indicate TA, where the models correctly identified unusual movements. (b) shows zoomed fish movements for months of 2020 with true anomalies.

At the top of each panel (a), a black square zoomed a specific region that focuses on a particular time frame in 2020. This region is examined in more detail in panel (b) of each figure, which zooms in on fish movements during this period. The zoomed-in view provides granular analysis of daily or monthly fish movements, enabling a closer evaluation of how each model performs in detecting anomalies. By narrowing the focus to this specific interval, the detailed visualization helps assess the models' precision and accuracy in identifying TA, offering insights into their reliability and effectiveness in capturing significant behavioural deviations.

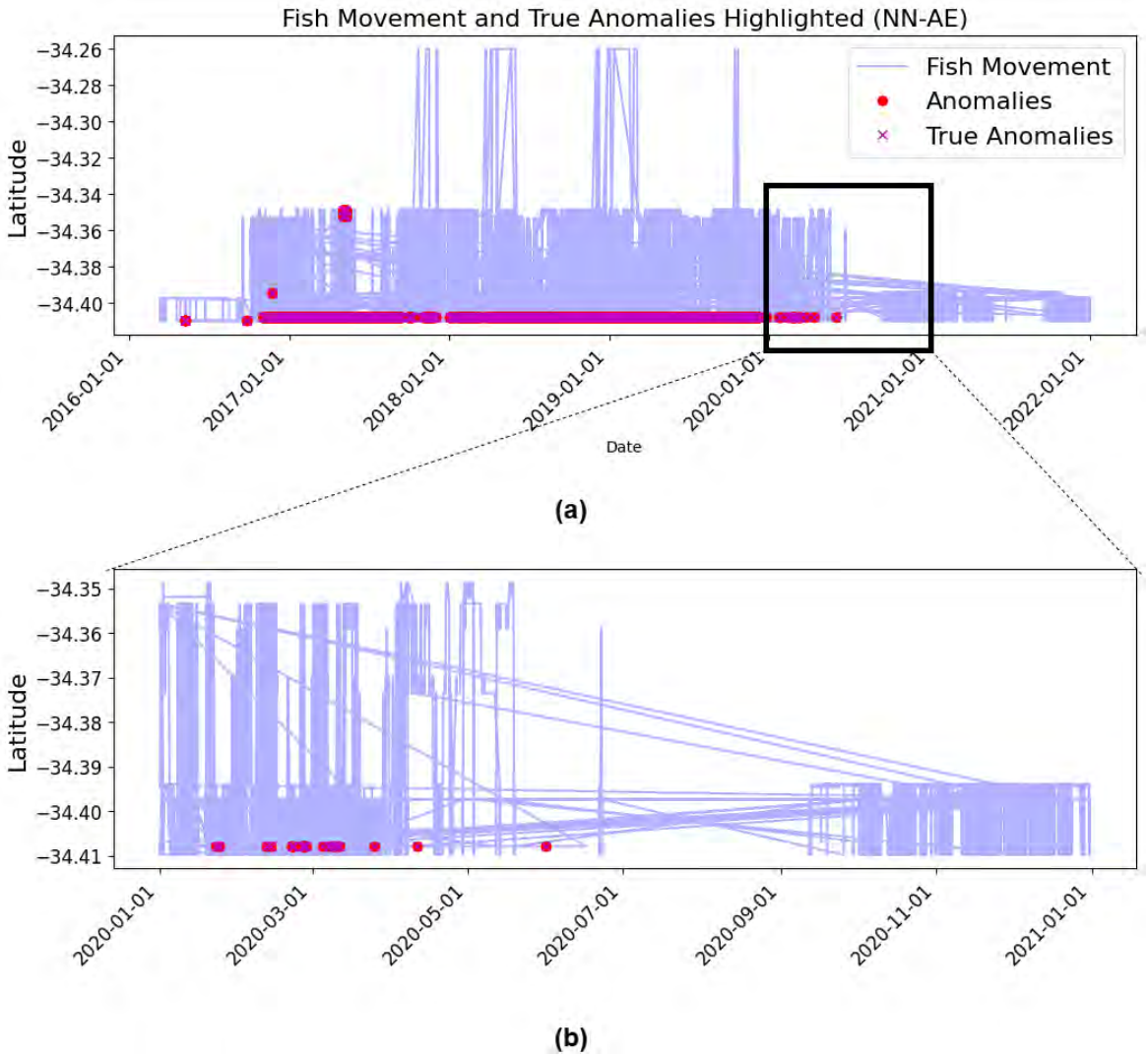


Figure 5.5: This figure shows the movement patterns of dusky kob in the Breede Estuary from 2016 to 2021, with a focus on highlighting anomalies and TA detected by the NN-AE. (a) An overview from 2016 to 2021, with blue lines indicating latitudinal movements, red dots highlighting anomalies, and magenta crosses marking TA. (b) A zoomed-in view of the 2020 movement patterns, shows a closer examination of anomalies and TA.

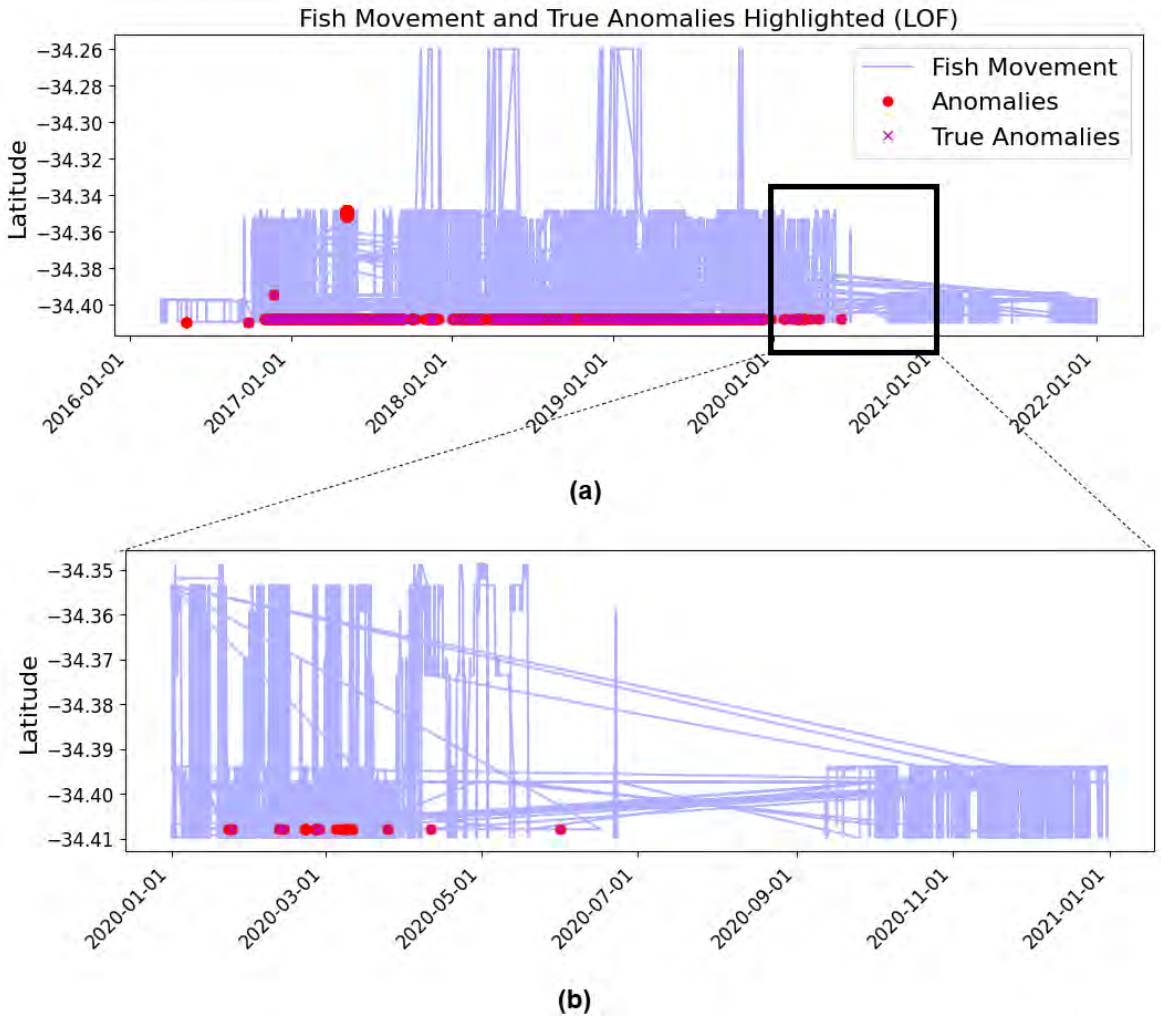


Figure 5.6: This figure shows the movement patterns of dusky kob in the Breede Estuary from 2016 to 2021, with a focus on highlighting anomalies and TA detected by the LOF. (a) An overview from 2016 to 2021, with blue lines indicating latitudinal movements, red dots highlighting anomalies, and magenta crosses marking TA. (b) A zoomed-in view of the 2020 movement patterns, shows a closer examination of anomalies and TA.

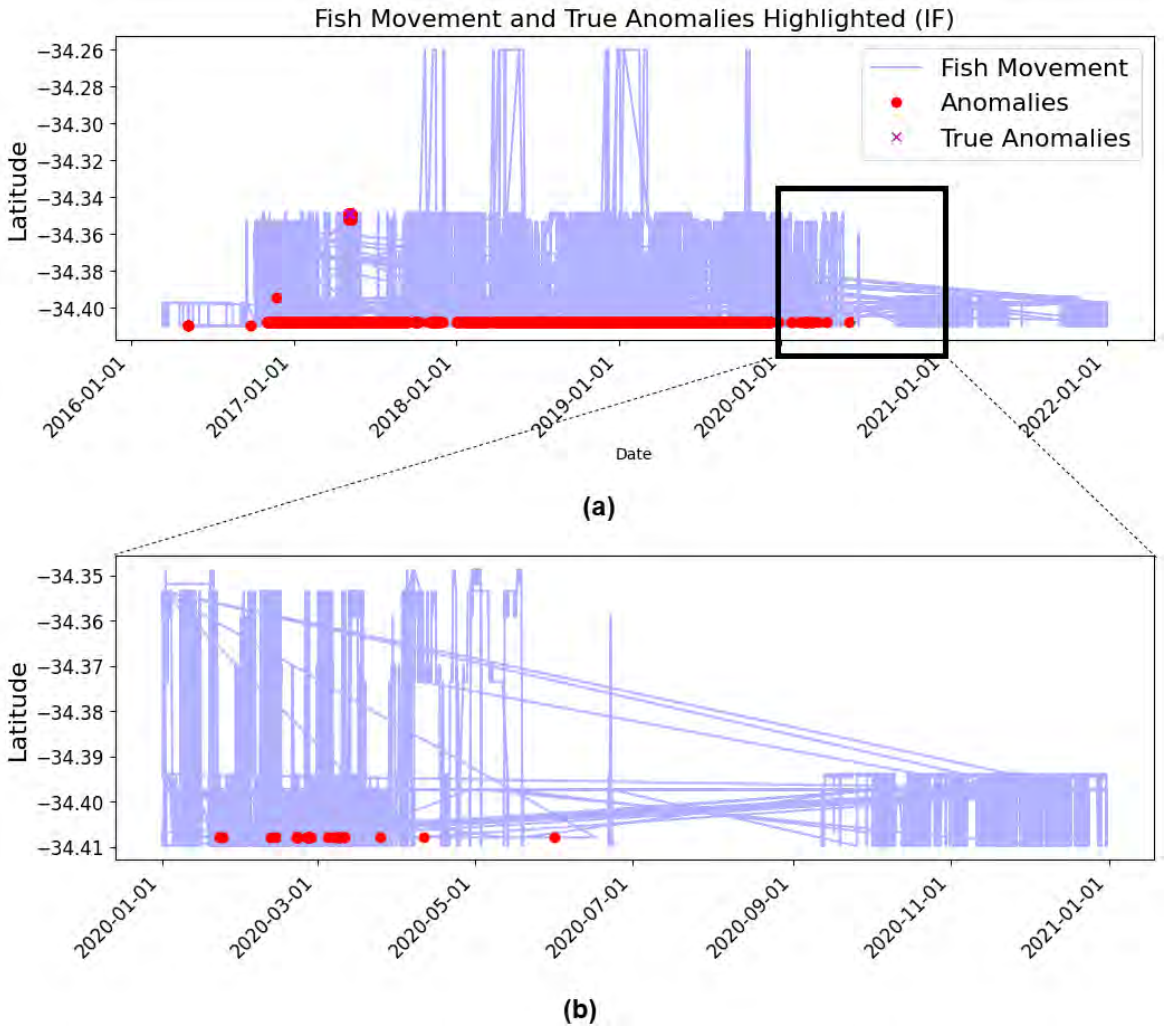


Figure 5.7: This figure shows the movement patterns of dusky kob in the Breede Estuary from 2016 to 2021, with a focus on highlighting anomalies and TA detected by the IF. (a) An overview from 2016 to 2021, with blue lines indicating latitudinal movements, red dots highlighting anomalies, and magenta crosses marking TA. (b) A zoomed-in view of the 2020 movement patterns, shows a closer examination of anomalies and TA.

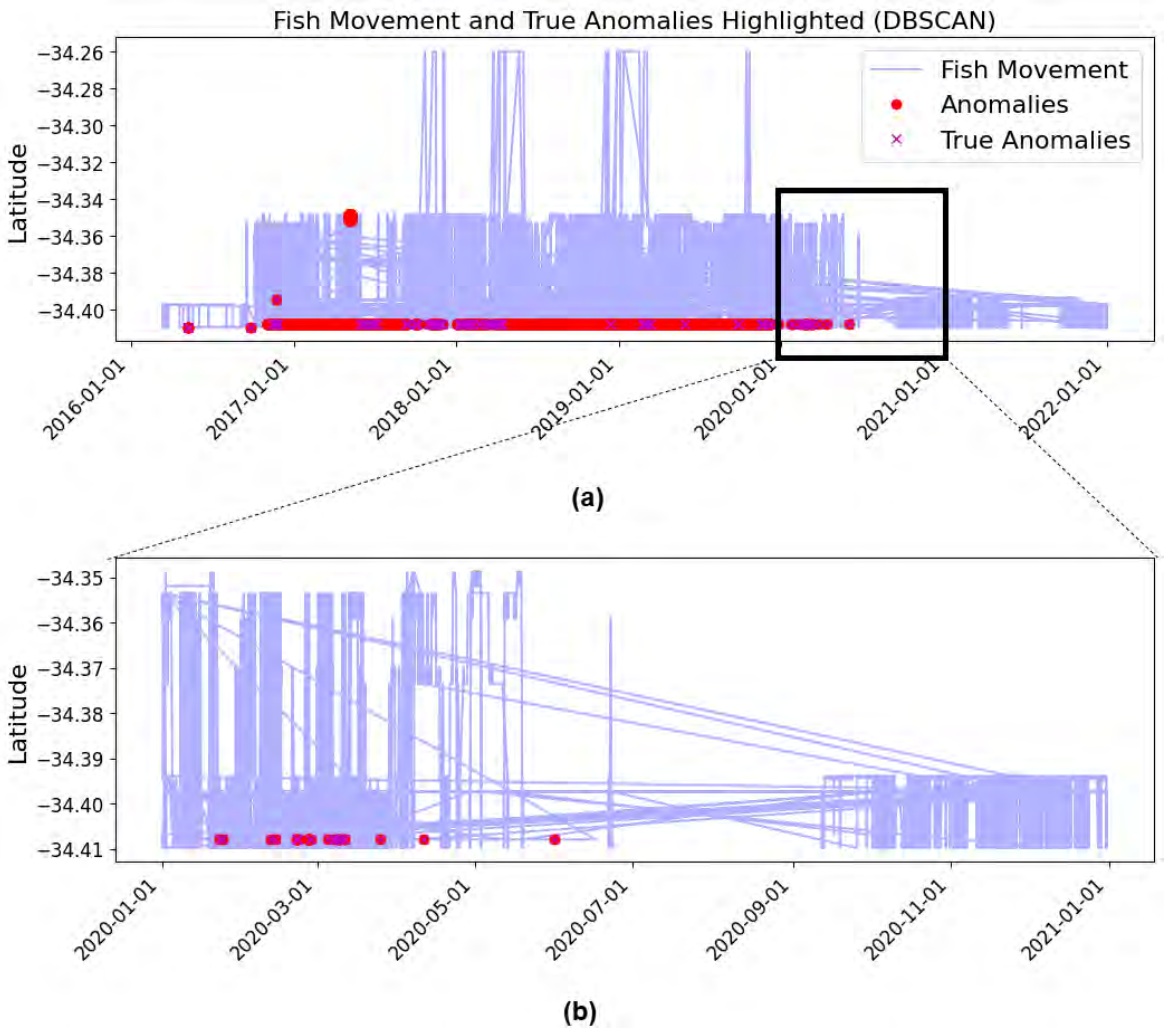
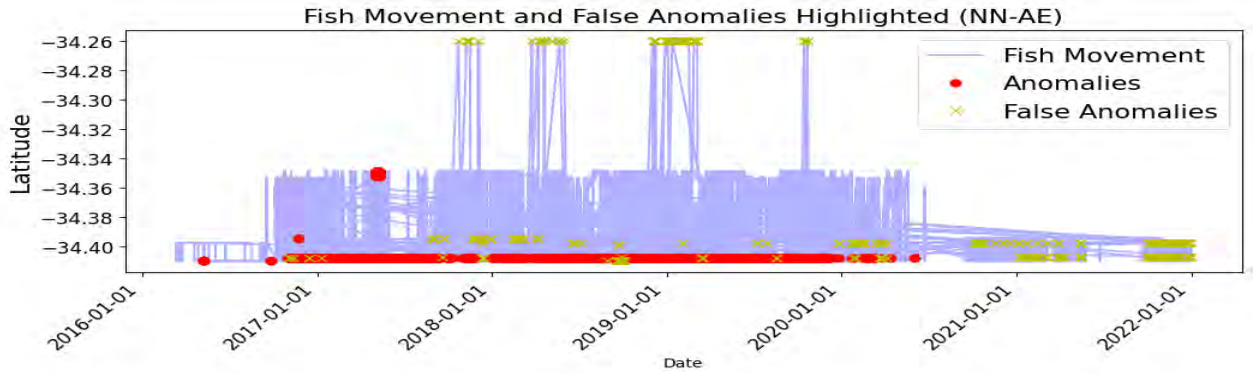


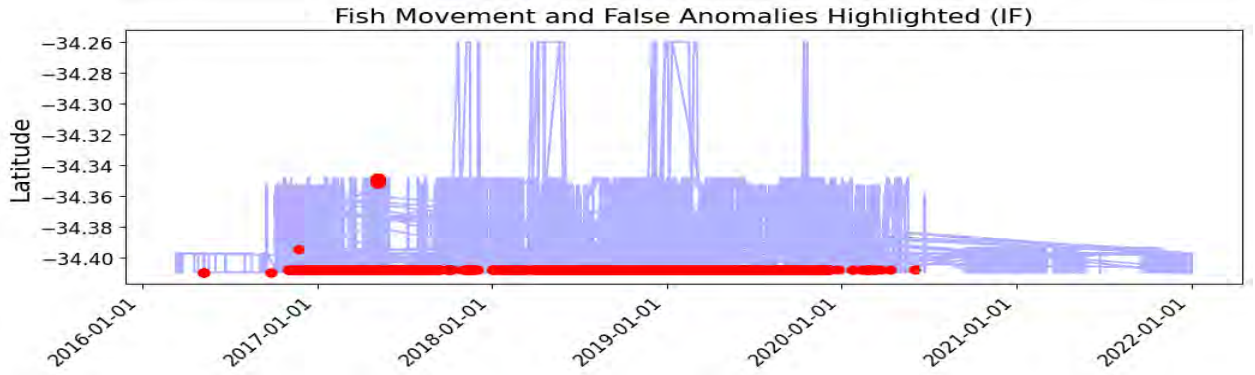
Figure 5.8: This figure shows the movement patterns of dusky kob in the Breede Estuary from 2016 to 2021, with a focus on highlighting anomalies and TA detected by the DBSCAN. (a) An overview from 2016 to 2021, with blue lines indicating latitudinal movements, red dots highlighting anomalies, and magenta crosses marking TA. (b) A zoomed-in view of the 2020 movement patterns, shows a closer examination of anomalies and TA.

Figure 5.9 compares the performance of the NN-AE, IF, DBSCAN, and LOF models in FA. Panel (b) of Figure 5.9 shows that the IF model did not produce any FA, while NN-AE generated the fewest FA among the remaining models, followed by LOF and DBSCAN. This indicates that IF was the most effective in avoiding the misclassification of normal behaviours as anomalies.

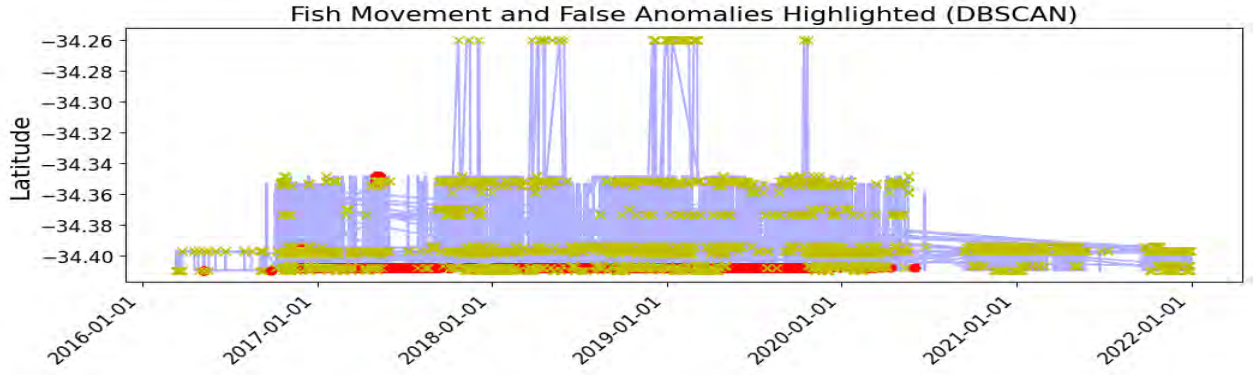
However, the absence of FA did not necessarily make IF the best overall model. While NN-AE produced slightly more FA than IF, it achieved 100% of TA with negligible FA. NN-AE's capacity to learn complex spatial and temporal patterns enabled it to identify meaningful anomalies that reflected significant changes in fish behaviour or environmental conditions, without over-representing irrelevant data. This capability positions NN-AE as the most suitable model for anomaly detection in telemetry data.



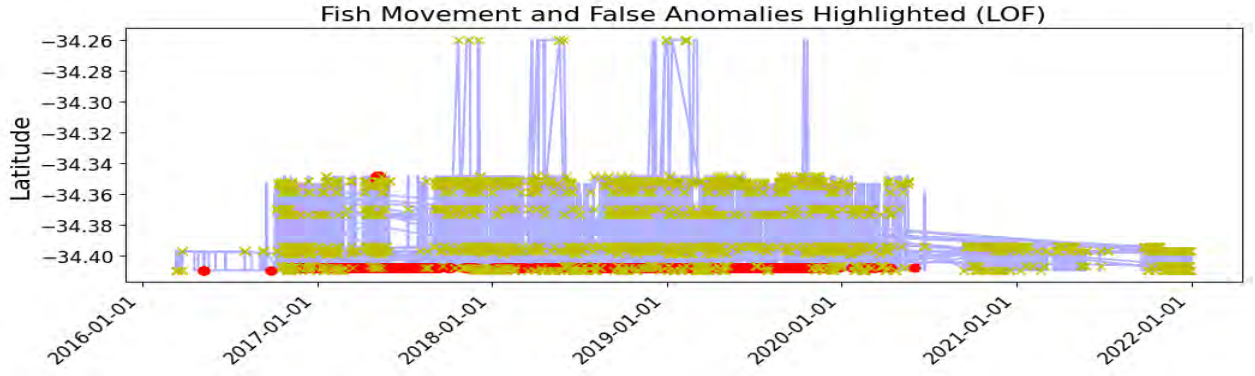
(a)



(b)



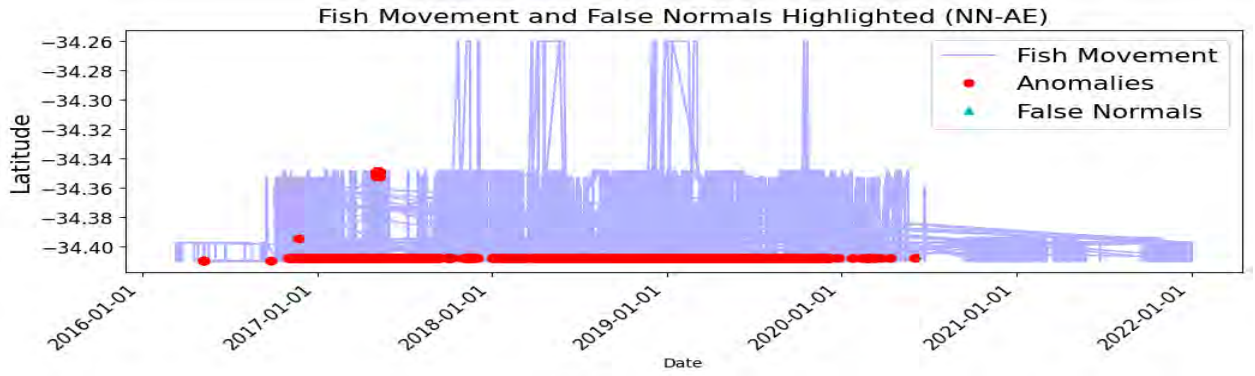
(c)



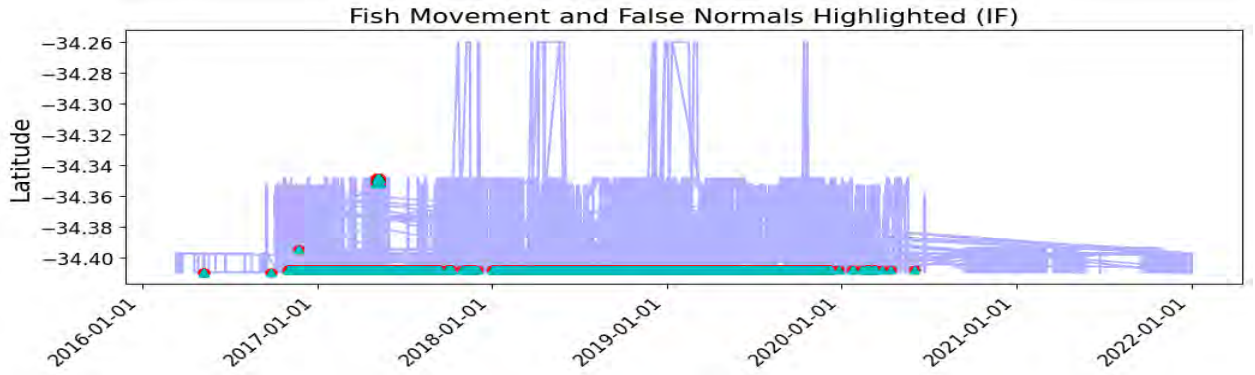
(d)

Figure 5.9: This figure shows the movement patterns of dusky kob in the Breede Estuary from 2016 to 2021, highlighting anomalies and FA detected by four machine learning models: NN-AE (a), IF (b), DBSCAN (c), and LOF (d). Blue lines represent fish movements, red dots indicate unusual movement patterns (anomalies) and magenta crosses mark correctly identified FA.

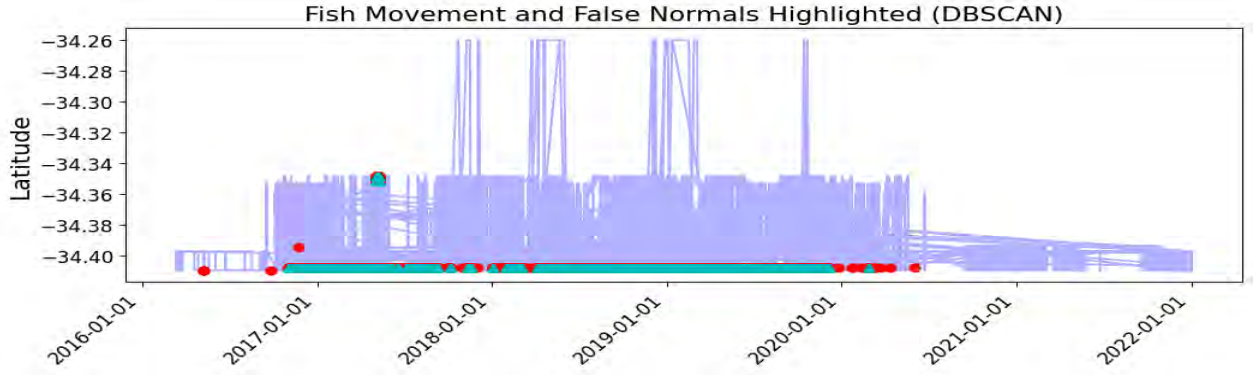
Figure 5.10 compares fish movement data over time, highlighting anomalies and FN detections. Each plot visualizes the changes in fish latitude from 2016 to 2021, with anomalies marked in red and FN in green. NN-AE model (a) achieved zero FN demonstrating its exceptional accuracy in detecting anomalies. This superior performance is attributed to the model's ability to learn complex temporal and spatial patterns in the fish movement data. The absence of FN indicated that NN-AE effectively identified all anomalies in the telemetry observation. This capability is particularly valuable for telemetry research, as removing anomalies could provide a more complete understanding of the factors influencing fish movement. In contrast, the IF (b), DBSCAN (c), and LOF (d) models misclassify a significant number of anomalies as normal instances, which undermines their effectiveness in detecting true behavioural anomalies. Figure 5.11 compares the performance of the NN-AE, IF, DBSCAN, and LOF models in detecting TN. All models show reasonable effectiveness in identifying TN, with the figure highlighting the IF model's strength in minimizing FA. However, its failure to detect true anomalies (TA) remains a critical limitation. This figure complements the results we got in Figure 5.1.



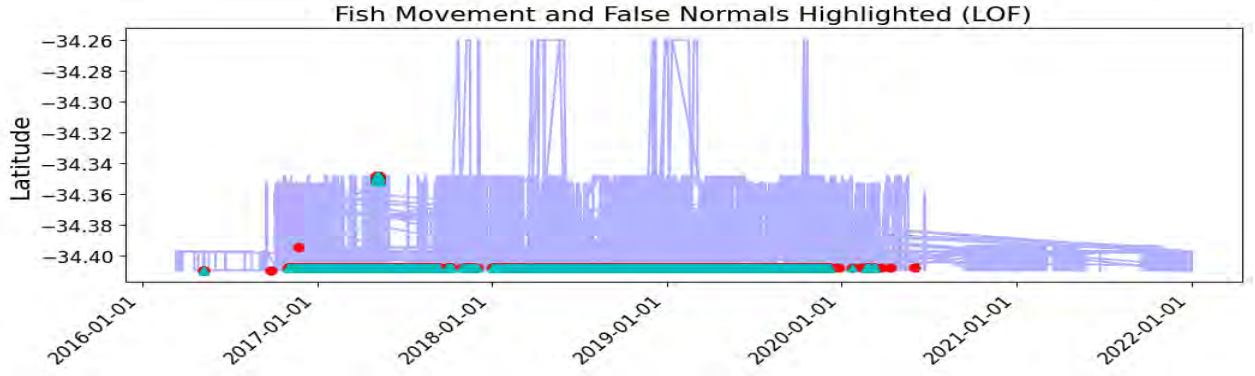
(a)



(b)

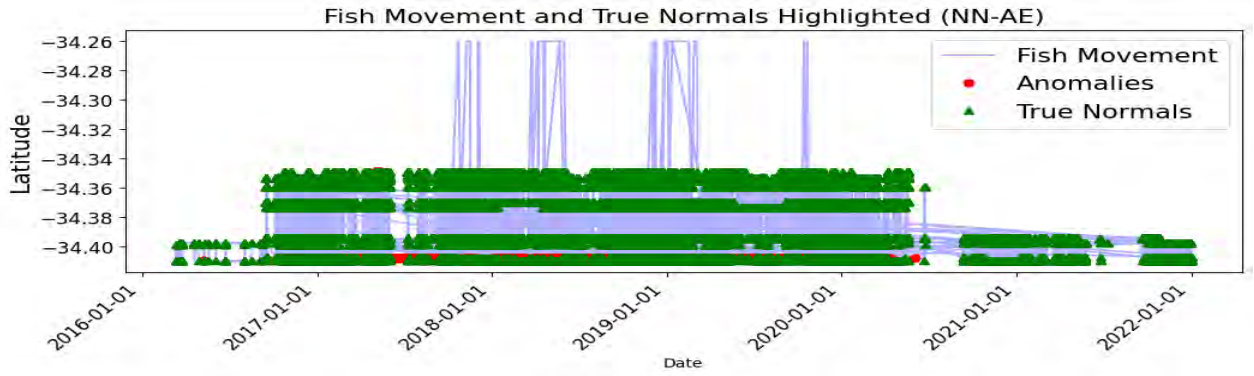


(c)

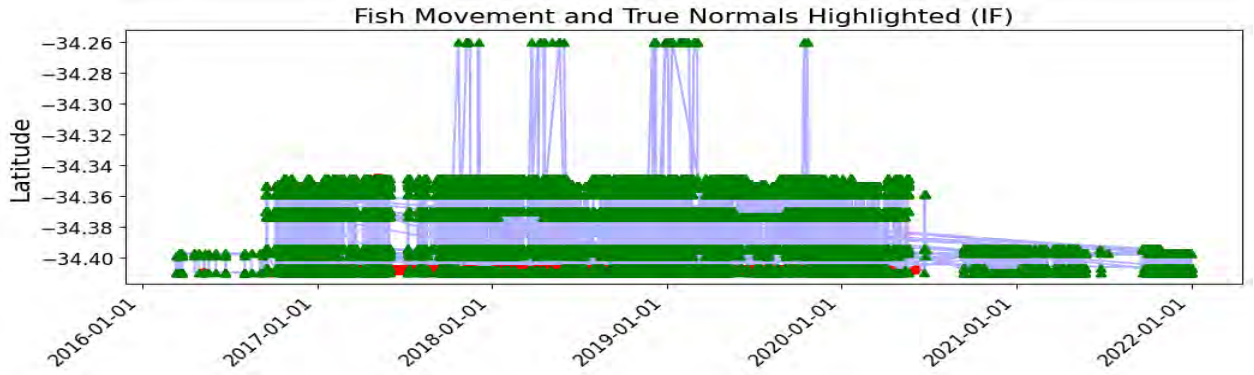


(d)

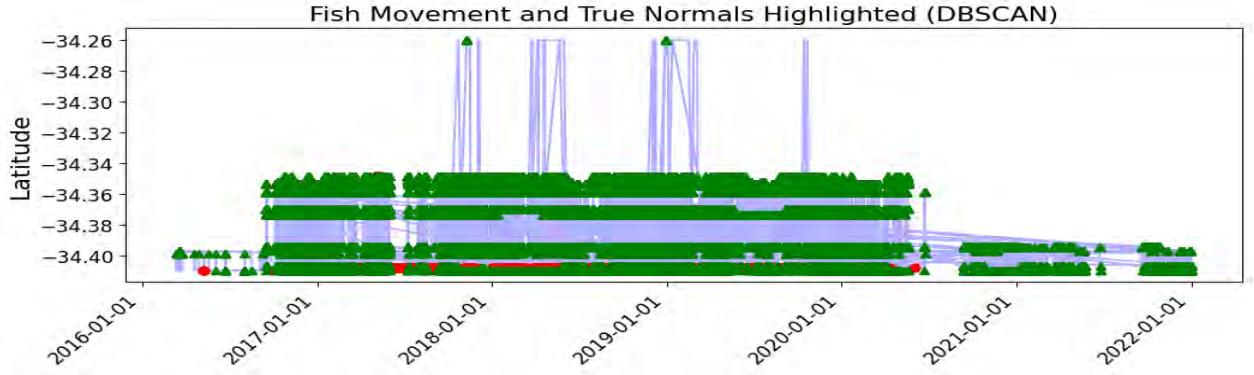
Figure 5.10: This figure shows the movement patterns of dusky kob in the Breede Estuary from 2016 to 2021, highlighting anomalies and FN detected by four machine learning models: NN-AE (a), IF (b), DBSCAN (c), and LOF (d). Blue lines represent fish movements, red dots indicate unusual movement patterns (anomalies) and magenta crosses mark correctly identified FN.



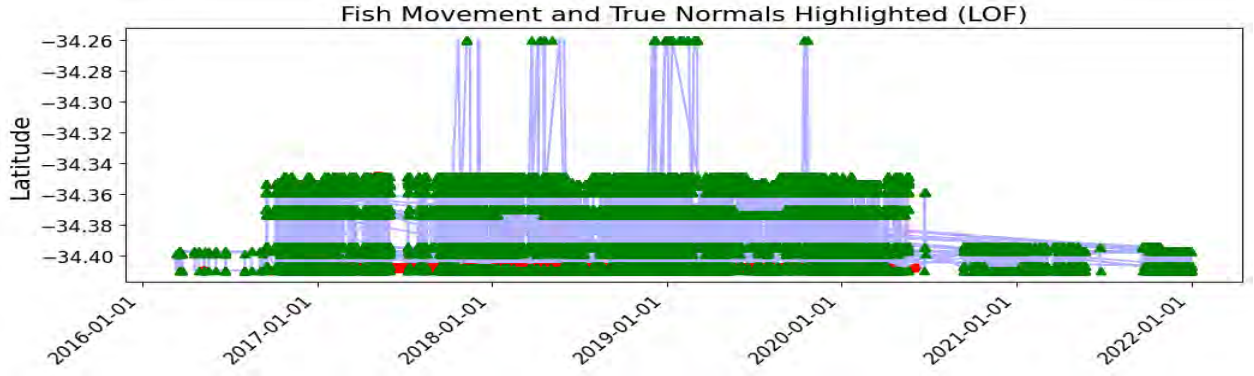
(a)



(b)



(c)



(d)

Figure 5.11: This figure shows the movement patterns of dusky kob in the Breede Estuary from 2016 to 2021, highlighting anomalies and TN detected by four machine learning models: NN-AE (a), IF (b), DBSCAN (c), and LOF (d). Blue lines represent fish movements, red dots indicate unusual movement patterns (anomalies) and magenta crosses mark correctly identified TN.

5.1.3 Interpretability of the NN-AE detection of FA and FN

Understanding the mechanisms behind NN-AE models' predictions of FA and FN provides valuable insights into the nature of anomalies and their correlations with normal data. Such an analysis can reveal patterns in telemetry datasets, helping to distinguish between true anomalies and noise or artifacts in the data. This is particularly relevant for the study of telemetry, where the boundary between normal and anomalous behaviour can be complex and context-dependent. There were three types of anomalies in the dataset: (i) fish recorded at only one station, (ii) normal/expected movements, but then remained stationary for more than 120 days, and (iii) a fish passing more than one consecutively positioned receiver. The NN-AE model correctly predicted all type 1 (259699 instances, 98.4% of anomalies) and type 3 (4271 instances, 1.6%) anomalies, with no false normals (FN = 0) as shown in Figure 5.2.

The NN-AE model showed strong performance in detecting anomalies in which fish were recorded at a single station throughout the study period, flagging all instances of this type of anomaly.

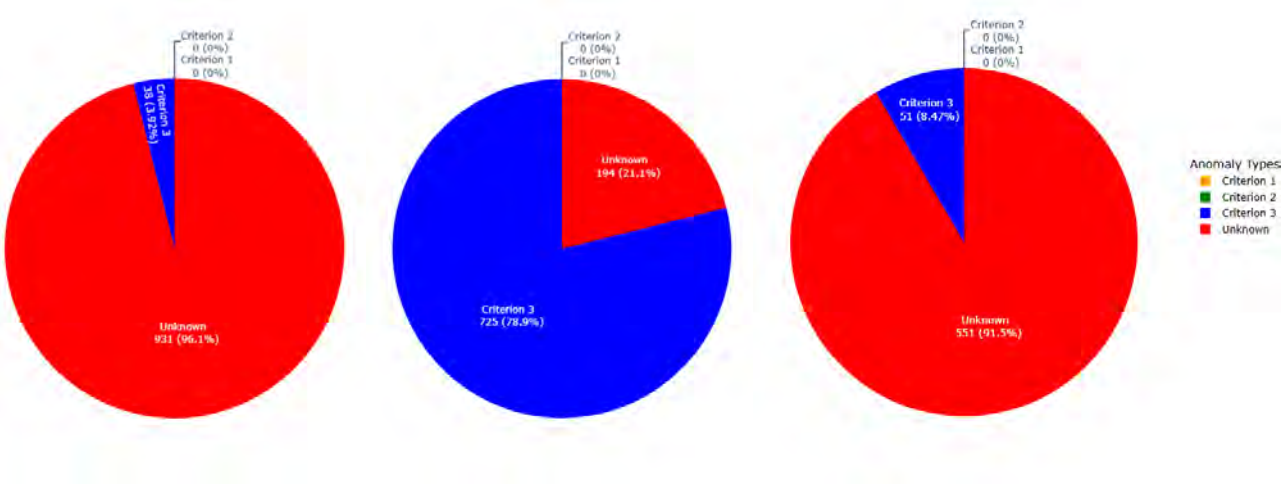
The second anomalous behaviour was not present in the dataset, preventing the NN-AE model from being tested on this criterion. To prepare the model for future scenarios where such behaviour might occur, synthetic data simulating such prolonged stays or expanding the dataset to capture more rare events would be beneficial. This would enhance the model's ability to detect and learn from extreme cases, improving its overall reliability in detecting anomalies.

The model effectively identified anomalies where fish were missed by more than one consecutive station. It flagged all true anomalies without any false normals, but some normal movement patterns with occasional detection gaps were incorrectly classified as anomalies (see Section 5.1.4 for a detailed discussion).

The models also misclassified as anomalies some normal detections that did not align with any of the three expert-defined criteria (see Section 5.1.4 for a detailed discussion), suggesting they occasionally flag unusual but not truly anomalous patterns. This over-sensitivity to movement pattern variations highlights a key challenge in automated anomaly detection for fish telemetry data: distinguishing between genuinely anomalous behaviour and natural behavioural variability. This limitation underscores the importance of incorporating biological knowledge and expert validation in the development and refinement of ML models for ecological applications. Future work could focus on training models with more diverse examples of normal movement patterns or implementing additional context-aware features that better account for natural behavioural variability.

5.1.4 False anomalies on the test dataset

We categorized all false anomalies identified by models trained and validated on the three datasets. The categorization process involved identifying false anomalies that resemble each of the three predefined anomaly criteria. Figure 5.12 shows the distribution of false anomalies for the datasets without resampling (a), with 90s resampling (b), and with 65s resampling (c), respectively. Red indicates false anomalies that do not resemble any predefined anomaly criteria (classified as unknown). Yellow are the false anomalies that resemble anomaly type *fish recorded at only one station* (Criterion 1), green are false anomalies that resemble anomaly of type *normal/expected movements, but then remained stationary for more than 120 days* (Criterion 2), and blue are false anomalies that resemble anomaly type *a fish passing more than one consecutively positioned receiver* (Criterion 3). The results show that some false anomalies closely resembled true anomalies falling under type *a fish passing more than one consecutively positioned receiver*, while others did not resemble any of the predefined criteria.



(a) **(b)** **(c)**

Figure 5.12: (a) False anomaly distribution for without sampling, (b) false anomaly distribution for 90s sampling interval, and (c) false anomaly distribution for 65s sampling interval.

5.1.5 Overall performance

NN-AE demonstrated an exceptional performance by showing no FN, a critical factor in ensuring that no true anomalies were missed. In contrast, IF, DBSCAN, and LOF exhibited significantly higher FN fraction of ~ 0.9989, ~ 0.9975, and ~ 0.9959, respectively, on the dataset without resampling. These high FN fractions indicated that these models failed to detect nearly all true anomalies, which is a considerable limitation, particularly in applications like telemetry observations where missing anomalies can impact interpretation of movement patterns. Moreover, these models did not show any meaningful improvement in reducing FN rates when resampling techniques were applied. This lack of responsiveness to resampling highlights their limited adaptability to variations in the data distribution and reinforces

their unsuitability for high-stakes anomaly detection tasks where the identification of true anomalies is paramount. The absence of FN in NN-AE underscores its robustness and reliability in anomaly detection tasks, making it a superior choice for datasets where accuracy in identifying anomalies is essential. This advantage is particularly pronounced in dynamic and large-scale datasets, such as those encountered in telemetry studies, where ensuring the complete identification of anomalies is critical for maintaining the integrity of the analysis.

In terms of FA, IF showed no FA detections, while NN-AE reports a negligible FA fraction of ~ 0.00352. This minimal FA rate in NN-AE further decreased when resampling techniques were applied, demonstrating its adaptability and capacity to improve its performance with data augmentation. On the other hand, DBSCAN and LOF exhibited FA fractions of approximately ~ 0.0242 and ~ 0.0128, respectively, on the dataset without resampling. Unlike NN-AE, these models showed no reduction in FA rates even with the application of resampling techniques, highlighting their limited ability to adapt to data augmentation.

The differences in FN and FA rates among these models underscore their varying capabilities and limitations in anomaly detection tasks. NN-AE's superior performance, marked by the absence of FN detections and negligible FA rates that further reduce with resampling, makes it particularly suitable for critical applications like telemetry, where both the identification of TA and the minimisation of FA are essential.

While this work demonstrates the impressive capabilities of the NN-AE model in accurately detecting anomalies, it also brings to light certain limitations that warrant further discussion. One significant challenge was the computational demand associated with training and optimizing the NN-AE model. Given the complexity of the architecture and the volume of data often encountered in telemetry and other large-scale applications, the model required substantial computational resources, which can be a limiting factor in environments with constrained infrastructure or limited access to high-performance computing. Another noteworthy limitation was the occurrence of false anomaly detections, where normal behaviour was misclassified as anomalous. Although the FA fraction ~ 0.00352 observed in this study was exceptionally low, such misclassifications, however negligible, may still have implications depending on the application. For instance, in critical domains like telemetry, monitoring even minimal false anomalies could lead to unnecessary interventions at the field. Nonetheless, it is worth emphasizing that the FA rate of the NN-AE model was significantly negligible compared to alternative models like DBSCAN or LOF, which exhibit much higher FA rates and failed to improve with resampling. The balance between computational efficiency and detection accuracy poses an ongoing trade-off in anomaly detection frameworks. While the NN-AE model offers a clear advantage in terms of Recall, Precision and adaptability, its computational intensity highlights the need for future research into more resource-efficient algorithms that retain comparable performance. Additionally, incorporating DL explainability mechanisms into the NN-AE model could enhance its

applicability by providing insights into the specific features or patterns leading to false anomaly detections. This could help refine the model further and reduce even the negligible rates of misclassification.

5.1.6 Summary

This chapter presents the comparative performance analysis of different anomaly detection models, including traditional approaches (IF, LOF, DBSCAN) and DL based methods (NN-AE). These models were evaluated using acoustic telemetry data of the dusky kob in the Breede Estuary, with key metrics such as accuracy, precision, recall, f_1-score, specificity and AUC guiding the evaluation. The NN-AE model showed superior performance on most evaluation metrics and was particularly excelling in its ability to detect TA without missing any FN. Unlike traditional models that struggled with high FN rates, NN-AE was able to effectively balance detection rates with minimal FA and further improved under resampling conditions. Temporal analyses provided insights into the reliability of the model over different time periods and highlighted the ability of NN-AE to detect subtle movement anomalies. Notably, traditional models such as IF showed competitive results in minimising FA, but fell short in accurately identifying TA.

Chapter 6

General Conclusion

This study focused on data collected from acoustically tagged dusky kob in the Breede Estuary, providing valuable insights into anomaly detection. The main problem addressed was the difficulty in detecting anomalies within these datasets, which often consist of millions of data points that are prone to errors. These anomalies, if they have not been dealt with, could skew ecological analyses and hinder effective conservation efforts. Thus, the motivation for this work lies in automating the anomaly detection in acoustic telemetry data to improve data integrity and enhance research efficiency.

The results demonstrated the effectiveness of the NN-AE model in anomaly detection. Accuracy, precision, and recall significantly outperformed traditional unsupervised learning models such as IF, LOF, and DBSCAN across all evaluation metrics, except for Specificity, where IF achieved the highest performance, followed closely by the NN-AE model. These results highlighted the NN-AE model's ability to capture complex temporal dependencies in time series data, enabling the detection of subtle anomalies that traditional methods often overlooked. Such capabilities were critical for telemetry studies and conservation efforts, where understanding anomalies in fish behavior was essential for assessing ecosystem dynamics, habitat usage, and responses to environmental changes.

To achieve these results, the study used a multi-step methodology described in earlier chapters. It introduced basic concepts of ML and DL and examined algorithms such as autoencoders and their applications in anomaly detection. The study provided an overview of the acoustic telemetry dataset, including pre-processing, standardization, and feature engineering to prepare the data for analysis. It also described the development and optimization of the NN-AE model and comparisons with traditional models such as IF, LOF, and DBSCAN. Finally, an in-depth evaluation of these models was conducted, highlighting the superior performance of the NN-AE model.

The practical implications of this work are substantial. By automating anomaly detection, the NN-AE model reduces the need for labour-intensive manual data review, enabling researchers

to focus on higher-level ecological analyses and conservation strategy development. In fisheries management, early detection of behavioural anomalies could alert stakeholders to threats such as habitat degradation, poaching, or disease outbreaks, facilitating timely interventions. This research provides a scalable framework that can be applied to other species and ecosystems, increasing its utility for global conservation efforts.

In conclusion, this study demonstrated the transformative potential of ML and DL, particularly NN-AE models, in automating anomaly detection in acoustic telemetry data. Beyond technical success, the ecological significance of this work is profound. Detecting movement anomalies provides crucial insights into environmental changes, human impacts, and fish health. This information is instrumental in developing conservation measures, such as identifying signs of habitat destruction, illegal fishing activities, or climate change effects on aquatic ecosystems. While the findings are directly applicable to dusky kob in the Breede Estuary, the methodology provides a scalable framework for studying other species and environments, contributing to global conservation efforts and advancing the role of AI in ecological research.

6.1 Limitations of this work

Despite these successes, certain limitations remain. The computational demands of training the NN-AE model remain a challenge, particularly in environments with limited access to high-performance computing resources. Additionally, while the false anomaly rate was exceptionally low, even minimal misclassifications can have implications in critical applications. The specificity of the findings to dusky kob in the Breede Estuary also limits their generalizability to other species, estuaries, or regions. Variations in environmental conditions, such as water temperature, salinity, or habitat structure, require recalibrating or retraining models when applied to new ecosystems. Furthermore, movement patterns, habitat preferences, and/or behaviours are often species-specific, and as such, each dataset may exhibit anomalies differently, requiring tailored feature engineering and anomaly detection techniques to maintain accuracy.

Although the NN-AE had a high anomaly detection recall rate, it is important to note that the anomaly dataset was heavily biased by a single individual (A69-9001-23722) that accounted for 98.4% of all anomaly detections due to its long stay at one station. This imbalance may have biased the performance of the model, as other types of anomalies (e.g. non-consecutive station visits or temporary prolonged stays) were underrepresented. Future analyses could address this imbalance by stratifying the anomaly classes or evaluating the model using subsets while excluding dominant outliers.

6.2 Future Work

This study also highlights areas for improvement and future exploration. Expanding the dataset to include a broader range of species and environmental conditions is critical to improving the generalizability and robustness of the model. Enhancing the model's ability to detect slow deviations from expected movement patterns remains a priority. Future work should address these limitations by exploring more computationally efficient algorithms and adaptive thresholding techniques that dynamically adjust to temporal data patterns. Expanding the dataset to include diverse species and environmental conditions will also enhance the robustness and generalizability of the model.

While the NN-AE achieved perfect recall (zero FN) and outperformed the traditional models in overall accuracy, the IF model completely eliminated FA. However, the almost complete inability of the IF to detect TA (FN rate ~ 0.9989) limits its utility as a standalone method. A promising direction for future research is to combine the strengths of both approaches, the high sensitivity of NN-AE for anomalies, and the strong specificity of IF for normal patterns through ensemble learning or hierarchical frameworks. For example, NN-AE could flag potential anomalies, which IF could then be used to reduce FA and thus optimize both recall and precision.

Furthermore, integrating attention-based mechanisms and explainability features into the NN-AE model could provide deeper insights into the factors contributing to anomalies, thereby improving its interpretability and utility. Exploring adaptive thresholds that dynamically adjust based on temporal data patterns could refine detection capabilities, particularly for subtle, gradual anomalies.

References

Ahmed, M., Mahmood, A. N., & Hu, J. (2016). A survey of network anomaly detection techniques. *Journal of Network and Computer Applications*, 60, 19–31.

Ahsan, M. & Salah, M. M. (2023). Efficient dcnn-lstm model for fault diagnosis of raw vibration signals: Applications to variable speed rotating machines and diverse fault depths datasets. *Symmetry*, 15(7), 1413.

AIML.com research (2024). Explain the basics architecture of neural networks, model training and key hyper-parameters. Accessed: 2024-10-09.

Al-Hamadani, M. N., Fadhel, M. A., Alzubaidi, L., & Harangi, B. (2024). Reinforcement learning algorithms and applications in healthcare and robotics: A comprehensive and systematic review. *Sensors*, 24(8), 2461.

Alowais, S. A., Alghamdi, S. S., Alsuhebany, N., Alqahtani, T., Alshaya, A. I., Almohareb, S. N., Aldairem, A., Alrashed, M., Bin Saleh, K., Badreldin, H. A., et al. (2023). Revolutionizing healthcare: the role of artificial intelligence in clinical practice. *BMC medical education*, 23(1), 689.

Benova, L. & Hudec, L. (2024). Comprehensive analysis and evaluation of anomalous user activity in web server logs. *Sensors*, 24(3), 746.

Berahmand, K., Daneshfar, F., Salehi, E. S., Li, Y., & Xu, Y. (2024). Autoencoders and their applications in machine learning: a survey. *Artificial Intelligence Review*, 57(2), 28.

Breunig, M. M., Kriegel, H.-P., Ng, R. T., & Sander, J. (2000). Lof: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data* (pp. 93–104).

Brima, Y. & Atemkeng, M. (2024). Saliency-driven explainable deep learning in medical imaging: bridging visual explainability and statistical quantitative analysis. *BioData Mining*, 17(1), 18.

Chambert, T., Miller, D. A. W., & Nichols, J. D. (2015). Modeling false positive detections in species occurence data under different study designs. *Ecology*, 96, 332–339.

Chen, H., Liu, H., Chu, X., Liu, Q., & Xue, D. (2021). Anomaly detection and critical scada parameters identification for wind turbines based on lstm-ae neural network. *Renewable Energy*, 172, 829–840.

Chen, Z., Yeo, C. K., Lee, B. S., & Lau, C. T. (2018). Autoencoder-based network anomaly detection. In *2018 Wireless telecommunications symposium (WTS)* (pp. 1–5).: IEEE.

Childs, A.-R. (2013). *Estuarine-dependency and multiple habitat use by dusky kob Argyrosomus japonicus (Pisces: Sciaenidae)*. PhD thesis, Rhodes University.

Childs, A.-R., Cowley, P. D., Næsje, T. F., & Bennett, R. H. (2015). Habitat connectivity and intra-population structure of an estuary-dependent fishery species. *Marine Ecology Progress Series*, 537, 233–245.

Cowley, P., Kerwath, S., Childs, A., Thorstad, E., Økland, F., & Næsje, T. (2008). Estuarine habitat use by juvenile dusky kob argyrosomus japonicus (sciaenidae), with implications for management. *African Journal of Marine Science*, 30(2), 247–253.

Cowley, P. D., Kerwath, S., Næsje, T., Childs, A.-R., Gillespie, L. D., Leggitt, C., Thorstad, E. B., & Økland, F. (2006). Space use patterns and movements of juvenile dusky kob argyrosomus japonicus in the great fish estuary (south africa): implications for management. *NINA rapport*.

Crossin, G. T., Heupel, M. R., Holbrook, C. M., Hussey, N. E., Lowerre-Barbieri, S. K., Nguyen, V. M., Raby, G. D., & Cooke, S. J. (2017). Acoustic telemetry and fisheries management. *Ecological Applications*, 27(4), 1031–1049.

Deng, L. (2012). The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE signal processing magazine*, 29(6), 141–142.

Dhellemmes, F., Aspillaga, E., & Monk, C. T. (2023). Atfiltr: A solution for managing and filtering detections from passive acoustic telemetry data. *MethodsX*, 10, 102222.

Ding, B., Qian, H., & Zhou, J. (2018). Activation functions and their characteristics in deep neural networks. In *2018 Chinese control and decision conference (CCDC)* (pp. 1836–1841).: IEEE.

Dufourq, E., Durbach, I., Hansford, J. P., Hoepfner, A., Ma, H., Bryant, J. V., Stender, C. S., Li, W., Liu, Z., Chen, Q., Zhou, Z., & Turvey, S. T. (2021). Automated detection of hainan gibbon calls for passive acoustic monitoring. *Remote Sensing in Ecology and Conservation*, 7(3), 475–487.

Elgindy, M. M., Elghamrawy, S. M., & El-Desouky, A. I. (2023). Proposed mitigation framework for the internet of insecure things. *Mansoura Engineering Journal*, 48(1), 3.

Ester, M., Kriegel, H.-P., Sander, J., Xu, X., et al. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96 (pp. 226–231).

Flávio, H., Baktoft, H., & Codling, E. (2021). actel: Standardised analysis of acoustic telemetry data from animals moving through receiver arrays. *Methods in Ecology & Evolution*, 12(1).

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Grant, G., Cowley, P., Bennett, R., Murray, T., & Whitfield, A. (2017). Space use by rhabdosargus holubi in a southern african estuary, with emphasis on fish movements and ecosystem connectivity. *African Journal of Marine Science*, 39(2), 135–143.

Griffin, L. P., Casselberry, G. A., Hart, K. M., Jordaan, A., Becker, S. L., Novak, A. J., DeAngelis, B. M., Pollock, C. G., Lundgren, I., Hillis-Starr, Z., Danylchuk, A. J., & Skomal, G. B. (2021). A novel framework to predict relative habitat selection in aquatic systems: applying machine learning and resource selection functions to acoustic telemetry data from multiple shark species. *Frontiers in Marine Science*, 8, 631262.

Griffiths, M. H. (1996). Life history of the dusky kob argyrosomus japonicus (sciaenidae) off the east coast of south africa. *South African Journal of Marine Science*, 17, 135–154.

Griffiths, M. H. & Heemstra, P. C. (1995). A contribution to the taxonomy of the marine fish genus argyrosomus (perciformes: Sciaenidae), with descriptions of two new species from southern africa. *Ichthyologica Bulletin JLB Smith Institute of Ichthyology*, 65, 1–40.

Guénard, G., Morin, J., Matte, P., Secretan, Y., Valiquette, E., & Mingelbier, M. (2020). Deep learning habitat modeling for moving organisms in rapidly changing estuarine environments: A case of two fishes. *Estuarine, Coastal and Shelf Science*, 238, 106713.

Habehh, H. & Gohel, S. (2021). Machine learning in healthcare. *Current genomics*, 22(4), 291.

Hamlomo, S. (2021). Statistical and mathematical learning: an application to fraud detection and prevention.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770–778).

Helm, J. M., Swiergosz, A. M., Haeberle, H. S., Karnuta, J. M., Schaffer, J. L., Krebs, V. E., Spitzer, A. I., & Ramkumar, P. N. (2020). Machine learning and artificial intelligence: definitions, applications, and future directions. *Current reviews in musculoskeletal medicine*, 13, 69–76.

Hindell, J. S., Jenkins, G. P., & Womersley, B. (2008). Habitat utilisation and movement of black bream acanthopagrus butcheri (sparidae) in an australian estuary. *Marine Ecology Progress Series*, 366, 219–229.

Hinton, G. E. & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *science*, 313(5786), 504–507.

Hitawala, S. (2018). Evaluating resnext model architecture for image classification. *arXiv preprint arXiv:1805.08700*.

Hong, W.-K. (2023). *Artificial Intelligence-Based Design of Reinforced Concrete Structures: Artificial Neural Networks for Engineering Applications*. Elsevier.

Howard, A. G. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 4700–4708).

Hughes, J. M., Meadows, N. M., Stewart, J., Booth, D. J., & Fowler, A. M. (2022). Movement patterns of an iconic recreational fish species, mulloway (argyrosomus japonicus), revealed by cooperative citizen-science tagging programs in coastal eastern australia. *Fisheries Research*, 247, 106179.

Hussey, N. E., Kessel, S. T., Aaerustrup, K., Cooke, S. J., Cowley, P. D., Fisk, A. T., Harcourt, R. G., Holland, K. N., Iverson, S. J., Kocik, J. F., Mills Flemming, J. E., & Whoriskey, F. G. (2015). Aquatic animal telemetry: a panoramic window into the underwater world. *Science*, 348, 1255642.

Ismael, O. (2025). The standard deviation score: a novel similarity metric for data analysis. *Journal of Big Data*, 12(1), 58.

Ismail, W. N., Alsalamah, H. A., Hassan, M. M., & Mohamed, E. (2023). Auto-har: An adaptive human activity recognition framework using an automated cnn architecture design. *Heliyon*, 9(2).

Jordan, M. I. & Mitchell, T. M. (2015). Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245), 255–260.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4, 237–285.

Kernbach, J. M. & Staartjes, V. E. (2022). Foundations of machine learning-based clinical prediction modeling: Part ii—generalization and overfitting. *Machine Learning in Clinical Neuroscience: Foundations and Applications*, (pp. 15–21).

Kerwath, S., Götz, A., Cowley, P., Sauer, W., & Attwood, C. (2005). A telemetry experiment on spotted grunter pomadasys commersonnii in an african estuary. *African Journal of Marine Science*, 27(2), 389–394.

Khan, A., Sohail, A., Zahoora, U., & Qureshi, A. S. (2020). A survey of the recent architectures of deep convolutional neural networks. *Artificial intelligence review*, 53, 5455–5516.

Klinard, Natalie, V., Matley, J. K., Ivanova, S. V., Larocque, S. M., Fisk, A. T., & Johnson, T. B. (2020). Application of machine learning to identify predators of stocked fish in lake ontario: using acoustic telemetry predation tags to inform management. *Journal of Fish Biology*, 98(1), 237–250.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278–2324.

Liu, F. T., Ting, K. M., & Zhou, Z.-H. (2008). Isolation forest. In *2008 eighth ieee international conference on data mining* (pp. 413–422).: IEEE.

Lowerre-Barbieri, S. K., Kays, R., Thorson, J. T., & Wikelski, M. (2019). The ocean's movescape: fisheries management in the bio-logging decade (2018-2028). *ICES Journal of Marine Science*, 76(2), 477–488.

Maleki, F., Muthukrishnan, N., Ovens, K., Reinhold, C., & Forghani, R. (2020). Machine learning algorithm validation: from essentials to advanced applications and implications for regulatory certification and deployment. *Neuroimaging Clinics*, 30(4), 433–445.

Maleki Varnosfaderani, S. & Forouzanfar, M. (2024). The role of ai in hospitals and clinics: transforming healthcare in the 21st century. *Bioengineering*, 11(4), 337.

Marciuc, A. M. (2021). Data preprocessing techniques. *BUILDING BRIDGES–INNOVATION, ENTREPRENEURSHIP AND SOCIETY*, (pp. 107).

Matley, J. K., Kilnard, N. V., Barbosa Martins, A. P., Aarestrup, K., Aspillaga, E., Cooke, S. J., Cowley, P. D., Heupel, M. R., Lowe, C. G., Lowerre-Barbieri, S. K., Mitamura, H., Moore, J.-S., Simpfendorfer, C. A., Stokesbury, M. J. W., Taylor, M. D., Thorstad, E. B., vandergoot, C. S., & Fisk, A. T. (2022). Global trends in aquatic animal tracking with acoustic telemetry. *Trends in Ecology and Evolution*, 37(1), 79–94.

Montesinos López, O. A., Montesinos López, A., & Crossa, J. (2022). *Multivariate statistical machine learning methods for genomic prediction*. Springer Nature.

Næsje, T. F., Cowley, P. D., Diserud, O. H., Childs, A.-R., Kerwath, S. E., & Thorstad, E. B. (2012). Riding the tide: estuarine movements of a sciaenid fish, argyrosomus japonicus. *Marine Ecology Progress Series*, 460, 221–232.

Nandutu, I. (2023). *Wildlife-Vehicle Collisions Mitigation Measures using Road Ecological Data and Deep Learning*. PhD thesis, RHODES UNIVERSITY.

Nasteski, V. (2017). An overview of the supervised machine learning methods. *Horizons. b*, 4(51-62), 56.

Ngai, E. W. & Wu, Y. (2022). Machine learning in marketing: A literature review, conceptual framework, and research agenda. *Journal of Business Research*, 145, 35–48.

Nhlapho, W., Atemkeng, M., Brima, Y., & Ndogmo, J.-C. (2024). Bridging the gap: Exploring interpretability in deep learning models for brain tumor detection and diagnosis from mri images. *Information*, 15(4), 182.

NICHD (2018). What are the parts of the nervous system? Accessed: 2024-12-05.

Niella, Y., Flávio, H., Smoothey, A. F., Aarestrup, K., Taylor, M. D., Peddemors, V. M., & Harcourt, R. (2020). Refined shortest paths (rsp): Incorporation of topography in space use estimation from node-based telemetry data. *Methods in Ecology and Evolution*, 11(12), 1733–1742.

Notte, D. V., Lennox, R. J., Hardie, D. C., & Crossin, G. T. (2022). Application of machine learning and acoustic predation tags to classify migration fate of atlantic salmon smolts. *Oecologia*, 198, 605–618.

Nwadiugwu, M. C. (2020). Neural networks, artificial intelligence and the computational brain. *arXiv preprint arXiv:2101.08635*.

R Core Team (2022). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

ramlakhan (2024). *Difference between Shallow and Deep Neural Networks*. Geek for Geeks.

Rayati, M., Sheikhi, A., & Ranjbar, A. M. (2015). Optimising operational cost of a smart energy hub, the reinforcement learning approach. *International Journal of Parallel, Emergent and Distributed Systems*, 30(4), 325–341.

Ruby, U. & Yendapalli, V. (2020). Binary cross entropy with deep learning technique for image classification. *Int. J. Adv. Trends Comput. Sci. Eng*, 9(10).

Sakabe, R. & Lyle, J. (2010). The influence of tidal cycles and freshwater inflow on the distribution and movement of an estuarine resident fish acanthopagrus butcheri. *Journal of Fish Biology*, 77(3), 643–660.

Sakurada, M. & Yairi, T. (2014). Anomaly detection using autoencoders with nonlinear dimensionality reduction. In *Proceedings of the MLSDA 2014 2nd workshop on machine learning for sensory data analysis* (pp. 4–11).

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 4510–4520).

Santhosh, A. & Idicula, S. M. (2017). Home range estimation of migrating organisms considering the spatiotemporal aspects of gps tracked data. In *2017 International Conference of Electronics, Communication and Aerospace Technology (ICECA)*, volume 1 (pp. 162–167).: IEEE.

Sarker, I. H. (2021). Deep learning: a comprehensive overview on techniques, taxonomy, applications and research directions. *SN computer science*, 2(6), 1–20.

Schindler, T. F., Schlicht, S., & Thoben, K.-D. (2023). Towards benchmarking for evaluating machine learning methods in detecting outliers in process datasets. *Computers*, 12(12), 253.

Sekkat, H., Moutik, O., Ourabah, L., ElKari, B., Chaibi, Y., & Tchakoucht, T. A. (2024). Review of reinforcement learning for robotic grasping: Analysis and recommendations. *Statistics, Optimization & Information Computing*, 12(2), 571–601.

Shah, V. (2020). Reinforcement learning for autonomous software agents: Recent advances and applications. *Revista Espanola de Documentacion Cientifica*, 14(1), 56–71.

Shanker, M., Hu, M. Y., & Hung, M. S. (1996). Effect of data standardization on neural network training. *Omega*, 24(4), 385–397.

Shanmuganathan, S. (2016). *Artificial neural network modelling: An introduction*. Springer.

Sharma, S., Sharma, S., & Athaiya, A. (2017). Activation functions in neural networks. *Towards Data Sci*, 6(12), 310–316.

Simonyan, K. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Simpfendorfer, C. A., Huveneers, Charlie; Steckenreuter, A., Tattersll, Katherine; Hoenner, X., Harcourt, R., & Heupel, M. R. (2015). Ghosts in the data: false detections in vemco pulse position modulation acoustic telemetry monitoring equipment. *Animal Biotelemetry*, 3(55), s40317–015–0094–z.

Spaulding, T. (2024). *telemetR: Filter and Analyze Generalised Telemetry Data from Organisms*. R package version 1.0.

Staudemeyer, R. C. & Morris, E. R. (2019). Understanding lstm–a tutorial into long short-term memory recurrent neural networks. *arXiv preprint arXiv:1909.09586*.

Sterratt, D., Graham, B., Gillies, A., Einevoll, G., & Willshaw, D. (2023). *Principles of computational modelling in neuroscience*. Cambridge University Press.

Szegedy, C., Ioffe, S., Vanhoucke, V., & Alemi, A. (2016). Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. *arXiv e-prints*, (pp. arXiv:1602.07261).

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1–9).

Ter Morhuizen, L. D., Whitfield, A. K., & Paterson, A. W. (1996). Influence of freshwater flow regime on fish assemblages in the great fish river and estuary. *South African Journal of Aquatic Sciences*, 22, 52–61.

Thorstad, E. B., Rikardsen, A. H., Alp, A., & okland, F. (2013). The use of electronic tags in fish research - an overview of fish telemetry methods. *Turkish Journal of Fisheries and Aquatic Sciences*, 13, 881–896.

Vakalopoulou, M., Christodoulidis, S., Burgos, N., Colliot, O., & Lepetit, V. (2023). Deep learning: basics and convolutional neural networks (cnns). *Machine learning for brain disorders*, (pp. 77–115).

Wang, J.-H., Lee, S.-K., Lai, Y.-C., Lin, C.-C., Wang, T.-Y., Lin, Y.-R., Hsu, T.-H., Huang, C.-W., & Chiang, C.-P. (2020). Anomalous behaviors detection for underwater fish using ai techniques. *IEEE access*, 8, 224372–224382.

Willmott, C. J. & Matsuura, K. (2005). Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance. *Climate research*, 30(1), 79–82.

Wu, Z., Zhang, C., Gu, X., Duporge, I., Hughey, L. F., Stabach, J. A., Skidmore, S. K., Hopcraft, J. G. C., Lee, S. J., Atkinson, P. M., McCauley, D. J., Lamprey, R., Ngene, S., & Wang, T. (2023). Deep learing enables satellite-based monitoring of large populations of terrestrial mammals across heterogenous landscape. *Nature Communications*, 14, 3072.

Xiang, D. (2024). Reinforcement learning in autonomous driving. *Applied and Computational Engineering*, 48, 17–23.

Ying, X. (2019). An overview of overfitting and its solutions. In *Journal of physics: Conference series*, volume 1168 (pp. 022022).: IOP Publishing.

Zahedi, L., Mohammadi, F. G., Rezapour, S., Ohland, M. W., & Amini, M. H. (2021). Search algorithms for automated hyper-parameter tuning. *arXiv preprint arXiv:2104.14677*.

Zhang, X., Huang, S., Zhang, X., Wang, W., Wang, Q., & Yang, D. (2018). Residual inception: a new module combining modified residual with inception to improve network performance. In *2018 25th IEEE International Conference on Image Processing (ICIP)* (pp. 3039–3043).: IEEE.

Zhu, L., Zhang, Z., Lin, G., Chen, P., Li, X., & Zhang, S. (2023). Detection and localization of tea bud based on improved yolov5s and 3d point cloud processing. *Agronomy*, 13(9), 2412.

Ziko, B., Murray, T., Næsje, T., Filmalter, J., & Cowley, P. (2023). Acoustic telemetry reveals the drivers behind estuary–sea connectivity of an important estuarine-dependent fishery species, pomadasys commersonnii, in the breede estuary, south africa. *African Journal of Marine Science*, 45(3), 201–213.

Appendix A

Tables A.1, A.2, and A.3 show the optimal thresholds for NN-AE trained and validated on three datasets: the original dataset without resampling, with 90s resampling, and with 65s resampling, respectively. The optimal percentile thresholds were 65 for the model trained on the dataset without resampling, 67 for the 90s resampling dataset, and 69 for the 65s resampling dataset. This trend suggests that datasets with higher temporal resolution (i.e., shorter resampling intervals) required slightly higher percentile thresholds for optimal anomaly detection.

Table A.1: Performance metrics (Precision, Recall, F_1-score, Specificity, and Accuracy) at different percentiles of reconstruction errors, demonstrating the trade-offs, and supporting the selection of the 65th percentile as the optimal threshold for anomaly detection in data without augmentation.

Percentile	Threshold	Precision	Recall	F_1-score	Specificity	Accuracy
1	0.007764	0.351275	1.0	0.519917	0.015156	0.357688
2	0.008336	0.354852	1.0	0.523824	0.030457	0.367667
3	0.008481	0.358488	1.0	0.527775	0.045702	0.377609
4	0.009629	0.362266	1.0	0.531858	0.061213	0.387726
5	0.009824	0.366179	1.0	0.536064	0.076946	0.397987
6	0.010049	0.370074	1.0	0.540225	0.092272	0.407982
7	0.011195	0.374071	1.0	0.544471	0.107670	0.418025
8	0.011339	0.378028	1.0	0.548650	0.122591	0.427757
9	0.011383	0.382179	1.0	0.553009	0.137913	0.437749
10	0.011512	0.386402	1.0	0.557417	0.153165	0.447697
11	0.011848	0.390943	1.0	0.562126	0.169194	0.458151
12	0.012492	0.395418	1.0	0.566738	0.184633	0.468220
13	0.012549	0.399941	1.0	0.571369	0.199885	0.478168
14	0.012802	0.404536	1.0	0.576042	0.215029	0.488044
15	0.013133	0.409309	1.0	0.580865	0.230403	0.498071
16	0.014166	0.414214	1.0	0.585786	0.245829	0.508132
17	0.014493	0.419164	1.0	0.590719	0.261033	0.518048
18	0.014778	0.424363	1.0	0.595864	0.276622	0.528215
19	0.015240	0.429626	1.0	0.601033	0.292016	0.538255
20	0.015895	0.435106	1.0	0.606375	0.307648	0.548450
21	0.016621	0.440711	1.0	0.611796	0.323236	0.558617
22	0.017105	0.446431	1.0	0.617286	0.338739	0.568728
23	0.018964	0.452512	1.0	0.623075	0.354792	0.579197
24	0.019656	0.458394	1.0	0.628629	0.369916	0.589061
25	0.020551	0.464440	1.0	0.634290	0.385059	0.598937
26	0.021317	0.470800	1.0	0.640196	0.400571	0.609054
27	0.021895	0.477062	1.0	0.645961	0.415439	0.618751
28	0.022122	0.483591	1.0	0.651919	0.430530	0.628594
29	0.023237	0.490313	1.0	0.658000	0.445650	0.638455
30	0.023745	0.497471	1.0	0.664415	0.461299	0.648661
31	0.024585	0.504572	1.0	0.670719	0.476386	0.658500
32	0.025161	0.511943	1.0	0.677199	0.491602	0.668424
33	0.025700	0.519722	1.0	0.683970	0.507194	0.678593
34	0.027399	0.527657	1.0	0.690806	0.522625	0.688657
35	0.029553	0.535921	1.0	0.697850	0.538209	0.698821
36	0.029990	0.544445	1.0	0.705037	0.553789	0.708982
37	0.030776	0.553183	1.0	0.712322	0.569259	0.719072
38	0.031713	0.562112	1.0	0.719682	0.584573	0.729060
39	0.032265	0.571391	1.0	0.727242	0.599979	0.739108
40	0.033151	0.581078	1.0	0.735041	0.615539	0.749256
41	0.034136	0.590770	1.0	0.742747	0.630594	0.759074
42	0.035481	0.600762	1.0	0.750595	0.645608	0.768867
43	0.037271	0.611319	1.0	0.758781	0.660937	0.778864
44	0.038495	0.622107	1.0	0.767036	0.676065	0.788730
45	0.038779	0.633444	1.0	0.775593	0.691406	0.798736
46	0.039302	0.644767	1.0	0.784022	0.706190	0.808378
47	0.041025	0.656769	1.0	0.792831	0.721306	0.818236
48	0.041799	0.669638	1.0	0.802136	0.736910	0.828413
49	0.043906	0.682682	1.0	0.811421	0.752126	0.838338
50	0.044223	0.696193	1.0	0.820889	0.767286	0.848225

Table A.1: Cont.

Percentile	Threshold	Precision	Recall	F_1-score	Specificity	Accuracy
51	0.044478	0.710544	1.0	0.830781	0.782757	0.858315
52	0.046398	0.725117	1.0	0.840658	0.797840	0.868152
53	0.049362	0.740752	1.0	0.851071	0.813363	0.878276
54	0.052883	0.756430	1.0	0.861327	0.828285	0.888008
55	0.053363	0.773131	1.0	0.872052	0.843513	0.897940
56	0.056369	0.790504	1.0	0.882996	0.858673	0.907827
57	0.060077	0.809227	1.0	0.894556	0.874281	0.918007
58	0.066410	0.828651	1.0	0.906297	0.889728	0.928081
59	0.081708	0.849439	1.0	0.918591	0.905478	0.938353
60	0.084467	0.869931	1.0	0.930442	0.920266	0.947998
61	0.098710	0.892116	1.0	0.942982	0.935510	0.957940
62	0.101801	0.915615	1.0	0.955949	0.950852	0.967946
63	0.117203	0.940660	1.0	0.969423	0.966359	0.978060
64	0.117517	0.966909	1.0	0.983176	0.981749	0.988097
65	0.120692	0.993855	1.0	0.996918	0.996703	0.997850
66	0.305324	1.0	0.977596	0.988671	1.0	0.992208
67	0.305671	1.0	0.948721	0.973686	1.0	0.982165
68	0.305921	1.0	0.920021	0.958345	1.0	0.972183
69	0.306270	1.0	0.891253	0.942500	1.0	0.962177
70	0.306705	1.0	0.862537	0.926196	1.0	0.952190
71	0.307026	1.0	0.833867	0.909409	1.0	0.942219
72	0.307326	1.0	0.805069	0.892009	1.0	0.932202
73	0.307718	1.0	0.776369	0.874108	1.0	0.922220
74	0.308059	1.0	0.747593	0.855568	1.0	0.912212
75	0.308449	1.0	0.718748	0.836363	1.0	0.902180
76	0.308957	1.0	0.689980	0.816554	1.0	0.892174
77	0.310452	1.0	0.661234	0.796076	1.0	0.882176
78	0.312662	1.0	0.632534	0.774911	1.0	0.872194
79	0.313046	1.0	0.603788	0.752953	1.0	0.862196
80	0.313664	1.0	0.575050	0.730199	1.0	0.852201
81	0.316172	1.0	0.546297	0.706587	1.0	0.842201
82	0.317073	1.0	0.517521	0.682061	1.0	0.832192
83	0.320032	1.0	0.488798	0.656634	1.0	0.822202
84	0.320922	1.0	0.466833	0.636519	1.0	0.814563
85	0.320926	1.0	0.432193	0.603540	1.0	0.802515
86	0.320967	1.0	0.402402	0.573875	1.0	0.792154
87	0.321086	1.0	0.373489	0.543855	1.0	0.782098
88	0.321163	1.0	0.345032	0.513046	1.0	0.772200
89	0.321251	1.0	0.316657	0.481002	1.0	0.762331
90	0.321349	1.0	0.287464	0.446559	1.0	0.752178
91	0.321493	1.0	0.258787	0.411169	1.0	0.742204
92	0.322034	1.0	0.230026	0.374018	1.0	0.732201
93	0.323296	1.0	0.201265	0.335089	1.0	0.722198
94	0.324078	1.0	0.172535	0.294294	1.0	0.712205
95	0.325454	1.0	0.143721	0.251322	1.0	0.702184
96	0.325955	1.0	0.115059	0.206372	1.0	0.692215
97	0.326316	1.0	0.086260	0.158820	1.0	0.682198
98	0.326627	1.0	0.057567	0.108867	1.0	0.672219
99	0.326864	1.0	0.028738	0.055871	1.0	0.662192
100	1.294590	0.0	0.0	0.0	1.0	0.652197

Table A.2: Performance metrics (precision, recall, f_1-score, specificity, and accuracy) at different percentiles of reconstruction errors, demonstrating the trade-offs, and supporting the selection of the 67th percentile as the optimal threshold for anomaly detection with augmented data using the 90s sampling rate.

Percentile	Threshold	Precision	Recall	F1-Score	Specificity	Accuracy
1	0.006183	0.328163	1.0	0.494161	0.014839	0.334894
2	0.007279	0.331551	1.0	0.497993	0.029824	0.345010
3	0.008931	0.334971	1.0	0.501840	0.044641	0.355014
4	0.009157	0.338433	1.0	0.505715	0.059334	0.364934
5	0.009793	0.342001	1.0	0.509688	0.074169	0.374949
6	0.010017	0.345617	1.0	0.513693	0.088892	0.384889
7	0.010216	0.349458	1.0	0.517923	0.104194	0.395219
8	0.010680	0.353281	1.0	0.522110	0.119094	0.405279
9	0.011087	0.357282	1.0	0.526467	0.134349	0.415578
10	0.011549	0.361237	1.0	0.530748	0.149093	0.425532
11	0.011866	0.365266	1.0	0.535084	0.163790	0.435454
12	0.011926	0.369464	1.0	0.539574	0.178757	0.445559
13	0.012397	0.373673	1.0	0.544049	0.193428	0.455464
14	0.012918	0.377979	1.0	0.548599	0.208099	0.465368
15	0.013253	0.382510	1.0	0.553356	0.223179	0.475549
16	0.013569	0.386962	1.0	0.557999	0.237653	0.485321
17	0.014060	0.391588	1.0	0.562793	0.252343	0.495238
18	0.014642	0.396336	1.0	0.567680	0.267065	0.505178
19	0.015152	0.401312	1.0	0.572766	0.282119	0.515341
20	0.015383	0.406249	1.0	0.577777	0.296692	0.525180
21	0.016471	0.411359	1.0	0.582926	0.311407	0.535114
22	0.016972	0.416781	1.0	0.588349	0.326625	0.545388
23	0.017516	0.422152	1.0	0.593681	0.341314	0.555305
24	0.018733	0.427626	1.0	0.599073	0.355905	0.565156
25	0.019276	0.433480	1.0	0.604794	0.371102	0.575415
26	0.019818	0.439228	1.0	0.610366	0.385631	0.585224
27	0.020752	0.445244	1.0	0.616151	0.400433	0.595218
28	0.021397	0.451311	1.0	0.621936	0.414962	0.605027
29	0.021789	0.457673	1.0	0.627950	0.429783	0.615033
30	0.022994	0.464117	1.0	0.633989	0.444381	0.624888
31	0.023402	0.470948	1.0	0.640332	0.459421	0.635042
32	0.023835	0.477815	1.0	0.646651	0.474107	0.644957
33	0.024215	0.484927	1.0	0.653132	0.488876	0.654928
34	0.024861	0.492325	1.0	0.659810	0.503788	0.664995
35	0.025308	0.499811	1.0	0.666498	0.518427	0.674878
36	0.026076	0.507678	1.0	0.673456	0.533346	0.684950
37	0.027332	0.515961	1.0	0.680705	0.548564	0.695225
38	0.027971	0.524266	1.0	0.687893	0.563337	0.705198
39	0.028229	0.532976	1.0	0.695348	0.578337	0.715325
40	0.028868	0.541850	1.0	0.702857	0.593124	0.725308
41	0.030642	0.551010	1.0	0.710517	0.607887	0.735275
42	0.032440	0.560660	1.0	0.718491	0.622919	0.745424
43	0.033150	0.570766	1.0	0.726736	0.638115	0.755683
44	0.034302	0.580823	1.0	0.734836	0.652714	0.765539
45	0.035420	0.591328	1.0	0.743188	0.667432	0.775475
46	0.036268	0.602303	1.0	0.751796	0.682260	0.785486
47	0.037056	0.613504	1.0	0.760462	0.696848	0.795335
48	0.037971	0.625625	1.0	0.769704	0.712044	0.805594
49	0.040528	0.638043	1.0	0.779031	0.727014	0.815701
50	0.043400	0.650496	1.0	0.788243	0.741452	0.825448

Table A.2: Cont.

Percentile	Threshold	Precision	Recall	F1-Score	Specificity	Accuracy
51	0.045826	0.663835	1.0	0.797958	0.756317	0.835483
52	0.048199	0.677659	1.0	0.807863	0.771104	0.845467
53	0.050456	0.692141	1.0	0.818065	0.785962	0.855498
54	0.052603	0.707160	1.0	0.828463	0.800728	0.865466
55	0.054454	0.722501	1.0	0.838897	0.815177	0.875221
56	0.059021	0.739263	1.0	0.850087	0.830278	0.885417
57	0.059345	0.756508	1.0	0.861378	0.845117	0.895435
58	0.061814	0.774042	1.0	0.872631	0.859526	0.905162
59	0.062266	0.792979	1.0	0.884538	0.874372	0.915185
60	0.063552	0.813056	1.0	0.896890	0.889357	0.925302
61	0.064624	0.834007	1.0	0.909492	0.904225	0.935340
62	0.066418	0.856123	1.0	0.922485	0.919129	0.945402
63	0.071635	0.879033	1.0	0.935623	0.933779	0.955293
64	0.080885	0.903258	1.0	0.949170	0.948461	0.965205
65	0.081862	0.928700	1.0	0.963032	0.963056	0.975058
66	0.083409	0.956018	1.0	0.977515	0.977862	0.985054
67	0.092559	0.984360	1.0	0.992119	0.992354	0.994838
68	0.247001	1.0	0.984945	0.992416	1.0	0.995109
69	0.247376	1.0	0.954177	0.976551	1.0	0.985113
70	0.247714	1.0	0.923385	0.960167	1.0	0.975110
71	0.248124	1.0	0.892670	0.943292	1.0	0.965131
72	0.248636	1.0	0.861909	0.925833	1.0	0.955137
73	0.249020	1.0	0.831072	0.907743	1.0	0.945119
74	0.249385	1.0	0.800288	0.889067	1.0	0.935118
75	0.249814	1.0	0.769550	0.869769	1.0	0.925132
76	0.250269	1.0	0.738720	0.849729	1.0	0.915117
77	0.250836	1.0	0.708035	0.829064	1.0	0.905148
78	0.251605	1.0	0.677221	0.807551	1.0	0.895137
79	0.253747	1.0	0.646392	0.785222	1.0	0.885121
80	0.255437	1.0	0.615608	0.762076	1.0	0.875120
81	0.256051	1.0	0.584832	0.738036	1.0	0.865122
82	0.258708	1.0	0.554063	0.713051	1.0	0.855126
83	0.259548	1.0	0.523325	0.687082	1.0	0.845140
84	0.261897	1.0	0.492457	0.659928	1.0	0.835112
85	0.263021	1.0	0.461757	0.631784	1.0	0.825138
86	0.263048	1.0	0.430829	0.602209	1.0	0.815090
87	0.263070	1.0	0.400455	0.571892	1.0	0.805222
88	0.263121	1.0	0.369103	0.539189	1.0	0.795037
89	0.263373	1.0	0.338592	0.505892	1.0	0.785124
90	0.263605	1.0	0.307823	0.470741	1.0	0.775128
91	0.263857	1.0	0.276766	0.433543	1.0	0.765039
92	0.264606	1.0	0.246240	0.395173	1.0	0.755122
93	0.265771	1.0	0.215487	0.354568	1.0	0.745131
94	0.266736	1.0	0.184703	0.311813	1.0	0.735130
95	0.268147	1.0	0.153896	0.266742	1.0	0.725121
96	0.268774	1.0	0.123112	0.219234	1.0	0.715121
97	0.269189	1.0	0.092306	0.169011	1.0	0.705112
98	0.269561	1.0	0.061568	0.115994	1.0	0.695126
99	0.269839	1.0	0.030708	0.059586	1.0	0.685101
100	1.137619	1.0	0.000030	0.000061	1.0	0.675071

Table A.3: Performance metrics (precision, recall, f_1-score, specificity, and accuracy) at different percentiles of reconstruction errors, demonstrating the trade-offs, and supporting the selection of the 69th percentile as the optimal threshold for anomaly detection with augmented data using the 65s sampling rate.

Percentile	Threshold	Precision	Recall	F1-Score	Specificity	Accuracy
1.0	0.006671	0.307389	1.0	0.470234	0.014377	0.314317
2.0	0.010666	0.310526	1.0	0.473895	0.028751	0.324317
3.0	0.012341	0.313727	1.0	0.477614	0.043125	0.334317
4.0	0.012690	0.316995	1.0	0.481391	0.057499	0.344317
5.0	0.014052	0.320332	1.0	0.485229	0.071873	0.354316
6.0	0.014491	0.323739	1.0	0.489129	0.086247	0.364315
7.0	0.015116	0.327220	1.0	0.493091	0.100621	0.374315
8.0	0.015249	0.330777	1.0	0.497119	0.114995	0.384315
9.0	0.016012	0.334412	1.0	0.501212	0.129369	0.394315
10.0	0.016358	0.338128	1.0	0.505375	0.143746	0.404317
11.0	0.016491	0.341927	1.0	0.509606	0.158120	0.414316
12.0	0.017035	0.345813	1.0	0.513909	0.172494	0.424316
13.0	0.017731	0.349788	1.0	0.518285	0.186868	0.434316
14.0	0.018500	0.353855	1.0	0.522737	0.201242	0.444316
15.0	0.019028	0.358018	1.0	0.527265	0.215615	0.454316
16.0	0.020337	0.362280	1.0	0.531873	0.229989	0.464315
17.0	0.020939	0.366644	1.0	0.536561	0.244363	0.474315
18.0	0.021852	0.371115	1.0	0.541334	0.258737	0.484315
19.0	0.022952	0.375698	1.0	0.546193	0.273114	0.494317
20.0	0.023814	0.380394	1.0	0.551139	0.287488	0.504316
21.0	0.024190	0.385209	1.0	0.556175	0.301862	0.514316
22.0	0.024612	0.390148	1.0	0.561304	0.316236	0.524316
23.0	0.024765	0.395214	1.0	0.566529	0.330610	0.534315
24.0	0.025395	0.400414	1.0	0.571851	0.344984	0.544316
25.0	0.026067	0.405753	1.0	0.577275	0.359358	0.554315
26.0	0.026479	0.411236	1.0	0.582803	0.373732	0.564315
27.0	0.027067	0.416869	1.0	0.588437	0.388106	0.574315
28.0	0.027698	0.422660	1.0	0.594183	0.402483	0.584316
29.0	0.028432	0.428613	1.0	0.600041	0.416857	0.594316
30.0	0.028545	0.434736	1.0	0.606015	0.431231	0.604316
31.0	0.028979	0.441036	1.0	0.612110	0.445605	0.614315
32.0	0.030082	0.447522	1.0	0.618328	0.459979	0.624315
33.0	0.030927	0.454201	1.0	0.624675	0.474353	0.634315
34.0	0.032055	0.461083	1.0	0.631152	0.488727	0.644315
35.0	0.033005	0.468176	1.0	0.637766	0.503100	0.654315
36.0	0.033521	0.475491	1.0	0.644519	0.517474	0.664315
37.0	0.033939	0.483040	1.0	0.651419	0.531852	0.674316
38.0	0.034578	0.490831	1.0	0.658466	0.546226	0.684316
39.0	0.036113	0.498877	1.0	0.665668	0.560599	0.694316
40.0	0.036358	0.507192	1.0	0.673029	0.574973	0.704316
41.0	0.036784	0.515788	1.0	0.680554	0.589347	0.714315
42.0	0.036888	0.524681	1.0	0.688250	0.603721	0.724315
43.0	0.037256	0.533885	1.0	0.696122	0.618095	0.734315
44.0	0.037454	0.543419	1.0	0.704175	0.632469	0.744315
45.0	0.037617	0.553299	1.0	0.712418	0.646843	0.754314
46.0	0.038178	0.563547	1.0	0.720857	0.661220	0.764316
47.0	0.039061	0.574180	1.0	0.729497	0.675594	0.774316
48.0	0.039714	0.585221	1.0	0.738347	0.689968	0.784316
49.0	0.040355	0.596696	1.0	0.747413	0.704342	0.794315
50.0	0.041110	0.608630	1.0	0.756706	0.718716	0.804315

Table A.3: Cont.

Percentile	Threshold	Precision	Recall	F1-Score	Specificity	Accuracy
51	0.041448	0.621665	1.0	0.766699	0.733786	0.814799
52	0.041923	0.634732	1.0	0.776558	0.748272	0.824877
53	0.042794	0.648075	1.0	0.786463	0.762461	0.834747
54	0.043785	0.662063	1.0	0.796677	0.776722	0.844669
55	0.045243	0.676877	1.0	0.807307	0.791182	0.854728
56	0.046965	0.692268	1.0	0.818154	0.805549	0.864724
57	0.047245	0.708431	1.0	0.829335	0.819966	0.874753
58	0.049649	0.725328	1.0	0.840800	0.834350	0.884760
59	0.053066	0.743171	1.0	0.852666	0.848830	0.894833
60	0.058090	0.761818	1.0	0.864809	0.863237	0.904856
61	0.065950	0.780976	1.0	0.877020	0.877323	0.914655
62	0.072420	0.801347	1.0	0.889720	0.891561	0.924561
63	0.075440	0.822972	1.0	0.902891	0.905905	0.934540
64	0.076202	0.845608	1.0	0.916346	0.920133	0.944438
65	0.077088	0.869489	1.0	0.930189	0.934341	0.954322
66	0.082129	0.895779	1.0	0.945025	0.949106	0.964594
67	0.084271	0.922321	1.0	0.959591	0.963159	0.974370
68	0.091506	0.950524	1.0	0.974634	0.977231	0.984160
69	0.098479	0.981761	1.0	0.990797	0.991873	0.994346
70	0.206688	1.0	0.985892	0.992896	1.0	0.995707
71	0.207152	1.0	0.952980	0.975924	1.0	0.985691
72	0.207554	1.0	0.920135	0.958406	1.0	0.975696
73	0.208043	1.0	0.887214	0.940237	1.0	0.965678
74	0.208644	1.0	0.854355	0.921458	1.0	0.955678
75	0.209169	1.0	0.821548	0.902033	1.0	0.945694
76	0.209650	1.0	0.788658	0.881843	1.0	0.935685
77	0.210060	1.0	0.755806	0.860922	1.0	0.925688
78	0.210598	1.0	0.722908	0.839172	1.0	0.915677
79	0.211333	1.0	0.690109	0.816644	1.0	0.905695
80	0.213318	1.0	0.657211	0.793153	1.0	0.895684
81	0.213755	1.0	0.624253	0.768665	1.0	0.885655
82	0.213862	1.0	0.591522	0.743341	1.0	0.875694
83	0.214154	1.0	0.558601	0.716798	1.0	0.865676
84	0.214395	1.0	0.525628	0.689064	1.0	0.855642
85	0.215216	1.0	0.492844	0.660275	1.0	0.845665
86	0.215768	1.0	0.460151	0.630278	1.0	0.835716
87	0.216072	1.0	0.427071	0.598528	1.0	0.825649
88	0.216426	1.0	0.394302	0.565591	1.0	0.815677
89	0.217883	1.0	0.361465	0.530995	1.0	0.805684
90	0.220101	1.0	0.328583	0.494637	1.0	0.795678
91	0.221149	1.0	0.295753	0.456496	1.0	0.785687
92	0.222492	1.0	0.262894	0.416335	1.0	0.775688
93	0.223228	1.0	0.230041	0.374038	1.0	0.765690
94	0.223757	1.0	0.197159	0.329378	1.0	0.755683
95	0.224262	1.0	0.164329	0.282273	1.0	0.745693
96	0.225074	1.0	0.131416	0.232304	1.0	0.735677
97	0.226338	1.0	0.098602	0.179505	1.0	0.725691
98	0.227912	1.0	0.065682	0.123267	1.0	0.715673
99	0.231056	1.0	0.032867	0.063643	1.0	0.705687
100	1.103892	0.0	0.0	0.0	1.0	0.695685