UiT The Arctic University of Norway

Faculty of Science and Technology
Department of Computer Science

Through space and time

Exploring the spatial and temporal dimensions in hydroacoustics and catch data

Erling Arvola Devold

INF-3981 - Master 023

Abstract

Modern fishing vessels have a wide range of instruments and sensors on board that are used for active fishing operations, with sonar equipment and echo sounders being among the most common. Sonar allows horizontal observation of the water column, while echo sounders provide more precise underwater environment monitoring. These instruments are useful as they are used today but require a lot of user experience for effective use. Estimating biomass density, fish size, and species is highly demanding, and the existing systems have significant uncertainties.

In this thesis, we propose a novel approach to hydroacoustic data analysis that capitalizes on catch reports as annotations for hydroacoustic transects. Combining catch messages with the positional attribute of echo data allows us to obtain annotated echo examples that describe the biota within a given location. The thesis leverages EchoBERT, a BERT-inspired model, as the underlying architecture.

To assess the capabilities of the annotations, we evaluate the model using different types of models. Both classification and regression tasks are employed, wherein the classification task aims to predict the presence of a species based on catch messages. In contrast, the regression tasks attempt to fit the model to the catch data and generate a distribution of the species.

Furthermore, we assess the model considering timestamps. Since the catch messages may not necessarily correspond to the same date as the echo data, we incorporate weighted loss functions that account for the temporal proximity. This approach allows for a closer association during the training process, where the outcome is weighted more heavily for temporally closer labels.

Our results provide insight into the characteristics of catch reports as annotations, shedding light on their usefulness and limitations. We also uncover potential bias present in the labelled data, where a seasonal fishing activity can be uncovered in the dataset. We also experiment and find the magnitude of difference in collation criterion when finding catch data based on the haversines formula.

Acknowledgements

I would like to extend my heartfelt appreciation to my advisors, Einar Holsbø and Peter Haro, for their invaluable guidance throughout the project. Their expertise and support have been instrumental in shaping the direction of my research.

I also want to sincerely thank Armin Pobitzer from SINTEF Ålesund and Bjørn Magnus Mathisen from SINTEF Digital for their valuable contributions and guidance within their respective domains.

I am truly grateful for the mentorship and support provided by all these individuals, which has significantly contributed to the successful completion of this work.

Finally, I would like to thank my dearest Ragnhild and my family for their support throughout.

Contents

List of Figures

List of Tables

List of Listings

1

Introduction

Species identification is a crucial aspect of fisheries research, as it enables the assessment of fish populations and supports sustainable fishing practices. Echosounders, which precisely monitor the underwater environment, have become a common tool in modern fishing fleets. These devices utilize hydroacoustic technology to gather data, allowing fishers to observe the presence and behaviour of fish beneath their boats.

Traditionally, categorising fish species observed in echograms has relied on manual interpretation by domain experts. However, this approach is subjective and requires specialized knowledge, leading to potential variations in findings. While acoustic target classification has the potential to estimate the abundance and biomass of marine ecosystems, there is currently no automated classification process available.

In 2005, the Directorate of Fisheries introduced an electronic reporting system for Norwegian fishing vessels operating in international waters. This system replaced the traditional method of faxing catch and activity reports and allowed for more efficient and accurate reporting. Subsequently, regulations were proposed to mandate reporting for all Norwegian vessels larger than 15 meters, further enhancing the documentation of sustainable fishing practices.

Supervised learning, a subfield of deep learning, has achieved significant success in various domains, including image, sound, and text classification tasks. However, successful supervised learning relies on well-defined datasets and

labelled examples for training models. In the context of hydroacoustic data analysis, using catch reports as annotations for hydroacoustic transects is a novel approach that offers cost-effective annotation compared to manual annotation by domain experts.

This research uses data from the sand eel survey conducted by the Norwegian Institute of Marine Research (HI) to validate the proposed annotation method with annotated hydroacoustic data. The potential impact of this system is substantial, as it can lead to advancements in abundance estimation. Fishing vessels and fisheries management can use this system as a decision support tool to validate their decisions and ensure sustainable practices.

By combining the power of deep learning and the availability of electronic catch reports, this research strives to develop an automatic classification process for hydroacoustic data. The successful implementation of this system can significantly improve species identification and abundance estimation and support effective fisheries management, ultimately contributing to the conservation and sustainability of marine ecosystems.

1.1 Problem definition

This thesis addresses the challenges of designing and implementing a deep learning pipeline for hydroacoustic data analysis. The pipeline incorporates multiple interconnected layers, including preprocessing and classification layers, which collaborate during inference. Our primary focus is on the training aspect of the pipeline, where we propose and evaluate novel approaches for labelling and training hydroacoustic data.

While this work does not delve into explainability and uncertainty quantification, we assess the model's performance based on relevant prior research [1, 2]. Therefore, our thesis is that:

Deep learning models applied to hydroacoustic data can be used to model fish abundance by using collated catch reportings as annotations

1.2 Targeted Applications

In this thesis, we develop a processing pipeline to infer and train deep-learning models for hydroacoustic data. The application holds great potential in various domains, including decision support systems and search engines, where his-

torical documentation and prediction of fishing locations based on parameters are crucial.

1.3 Assumptions and Limitations

The thesis main focus is the annotation method, and experiments to evaluate the annotation in a deep learning architecture. In order to fulfill the requirements, we constrained the implementation by reducing complexities in logic. These are:

- We will assume the catch data to be static, and will not implement mechanisms for periodically fetching new .csv files.
- We will only look at one of the frequencies in the hydroacoustic data.
- We will limit the scope of the experiments to a specific deep learning architecture, and will not consider alternative architectures or compare performance across different models.
- The evaluation of the annotation method will be conducted on a specific dataset, and the results may not generalize to other datasets or domains.
- The experiments will be conducted on a specific hardware setup and may not reflect the performance of the annotation method on different hardware configurations.

These assumptions and limitations help to streamline the implementation and focus on the core objectives of the thesis while acknowledging the potential constraints and factors that may influence the results and generalizability of the proposed annotation method.

1.4 Methodology

With the rapid expansion of the computer science discipline, there arose a need for a comprehensive understanding and definition of the field. The *ACM's Task Force of the Core of Computer Science* was in 1989, formed to charter this discussion. The task force introduced three paradigms. [3] The first paradigm, *Theory*, is deeply rooted in mathematics and involves the development of coher-

ent and valid theories. It follows a four-step process: characterizing the objects of study, hypothesizing possible relationships among them, determining the truth of these relationships through proof, and evaluating and interpreting the results obtained. This paradigm emphasizes rigorous mathematical reasoning and iterative refinement to address errors and inconsistencies.

The second paradigm, *Abstraction*, draws inspiration from experimental scientific methods. It is an iterative process that aims to examine phenomena. The four steps involved in this paradigm are: forming a hypothesis, constructing a model to make predictions based on the hypothesis, designing an experiment to collect relevant data, and analyzing the results obtained. Abstraction allows for exploring complex systems by simplifying them into manageable models.

The third paradigm, *Design*, is rooted in engineering principles and focuses on the iterative creation of systems. It involves four steps: stating requirements, specifying the system based on those requirements, designing and implementing the system, and testing its functionality. This paradigm emphasizes applying computer science principles to build real-world solutions.

In the thesis at hand, the research is situated within the framework of Design Science in Information Systems Research [4], which incorporates the three paradigms introduced by the ACM's Task Force. The initial stages of the thesis involved collecting existing knowledge and theories to establish a solid theoretical foundation. With this knowledge, requirements for the system were defined, and components were designed to meet the specifications derived from those requirements. We were able to implement the system for labelling and classification from the theory and abstractions, and by following the iterative process, we successively increased our knowledge of the domain.

By obtaining more knowledge through the iterative process, we discovered new and refined existing requirements, enabling us to implement functionality to satisfy these. Finally, we conducted an experimental evaluation of the system. The evaluation aimed to demonstrate the system's capabilities and feasibility using quantitative methods. Through this evaluation, we provided evidence of the system's performance and its alignment with the defined requirements. Using quantitative methods allowed for a rigorous assessment of the system's effectiveness and providing measurable results.

1.5 Context

This thesis is written as part of the Norwegian Seafood Research Fund project **Datafangst**[5] in collaboration with my advisors *Peter Haro* (SINTEF Nord), *Einar Holsbø* (Arctic University of Tromsø).

1.6 Contributions

The contributions of this work are:

- Models for
  - Labelled echo data models to quickly index features.
  - Annotation of echo data with catch messages.
  - Pytorch-specific data loader for echo data.
- Methods for
  - Collection of catch messages from sequential data stores.
  - Processing of echo data from Kongsberg .raw files.
  - Collation of data sources by positional attributes.
  - Noise filtering and seabed cropping of echo data.
  - Calculation of temporal-proximity weighting of loss functions
- Artifacts
  - Pipeline for echo and catch data.

1.7 Thesis Outline

The rest of the thesis is structured as follows:

Chapter 2: Background explores the necessary theories and background for

understanding the premise of this thesis, as well as related works. The chapter introduces Hydroacoustics, image processing techniques, and Deep Learning.

Chapter 3: Related Work provides a review of related literature and "state-of-the-art" methods for deep learning and echo data.

Chapter 4: Design presents the requirements for the pipeline and its design.

Chapter 5: Implementation describes the implementation of the design, focusing on the methods used to fulfill the requirements.

Chapter 6: Evaluation outlines the experiments and the results of the annotation method and deep learning model.

Chapter 7: Discussion discusses the choices made in the pipeline's design and their impact on performance, feasibility, and practicality.

Chapter 8: Conclusion summarizes the thesis and proposes new fundamental ideas for future work.

2 Background

This chapter provides an overview of theories and technologies needed to understand the basis of related work and the thesis. Moreover, this chapter will cover hydroacoustics regarding echo sounders and data processing. We will also describe Deep learning and relevant architectures of state-of-the-art models.

2.1 Hydroacoustics

Modern fishing boats are equipped with a wide range of instruments and sensors that are used for fishing operations, with echo sounders and sonar being the most common. Sonar is used to look horizontally through the water, and echo sounders provide higher precision of the environment under the vessel. In this thesis, we delimit us to echo sounders.

Echosounders collect data by transmitting sound through the water column, bouncing off objects, *biota* or seabed. The time it takes for the sound to return determines the distance between the echosounder and the object, while the strength and frequency of the sound provide information about the object's characteristics, such as size and composition.

When the echosounder transmits sound pulses over a period of time, this results in a two-dimensional picture, or echogram, of targets over time. The horizontal

extent of targets shows changes in time if the echosounder is stationary or in space if the echosounder is on a moving vessel. The vertical extent of targets indicates the height of the object, such as a fish school or scattering layer.

Understanding echo-sounding data requires domain expertise and can be a difficult and tedious task to do manually.

2.2 Stabilizing hydroacoustic data

Raw hydro-acoustic data requires processing to improve the quality of the data source. The response of the hydro-acoustic sensor is often noisy, with unwanted frequencies and artefacts, from the sensor itself or other sources. This section will cover theories for removing unwanted features and noise.

2.2.1 Spatial filters

A fundamental part of image processing is filtering techniques, which involve modifying the pixels of an image based on a kernel. The kernel defines how each pixel value is combined with the values of its neighbouring pixels to produce a modified pixel.

There are two main categories of image filters, namely spatial domain filters and frequency domain filters. The frequency domain filters revolve around removing, modifying or passing specified frequency components of an image. Spatial filtering pertains to modifying an image by replacing the value of each pixel with a function of the pixel itself and its neighbours. The operation or mathematical function can be linear or nonlinear.

The spatial filters can blur or sharpen an image. A linear spatial filter, such as a low pass filter, reduces sharp intensities in images. Noise often comes as sharp transitions in intensity, and applying a low pass filter yields an adequate application in noise reduction[6].

Order-statistic filters are nonlinear spatial filters with responses based on ranking the pixels in the kernel region. Smoothing is achieved by replacing the value of the centre pixel with the value determined by the rank result. The kernel evaluates each pixel in the region and ranks each corresponding pixel's intensity. The pixel chosen is based on the filter criteria. The best-known order-statistic filter, the median filter, chooses the median pixel as its response and the centre pixel is replaced with the response.

2.2.2 Frequency filters

Frequency filters allow filtering by modifying, removing or adding frequencies in the image spectrum. Fourier Transform models the signal in the frequency domain, and by modifying the spectrum, the inverse transform will obtain the processed result.

These filters can come in handy when periodic noise exists in the data. In the context of hydroacoustics, sounds from other bodies, such as the boat, can give frequencies that can be seen as periodic noise. By observing and eliminating the said noise frequency, the echogram can be retrieved without noise.

Image smoothing is one application for frequency domain filters. This is done by *low pass filtering* for high-frequency attenuation. Ideal, Butterworth and Gaussian filters are three types of low-pass filters, each with its own distinct properties in the response. The ideal lowpass is very sharp, whereas Gaussian filters are smooth. The Butterworth filter is a combination of these two extremes, where the assumption is that the higher filter order will approach the response of an ideal filter, and with a smaller filter order, the Butterworth filter approaches a Gaussian filter[7].

2.3 Deep learning

This section describes the foundation for theories in deep learning that are applied in related work and for the models introduced in this thesis.

Machine learning can be divided into two main categories, supervised and unsupervised learning. Supervised learning algorithms involve training a model on labelled data with input-output pairings. The goal is to learn an objective function to infer output for new, unseen input data. unsupervised...

Deep learning is a subfield of machine learning that learns a hierarchical representation of data. The difference between machine learning and deep learning lies in the aspect of the multiple layers of neural networks found in deep learning. There exist multitudes of unique neural networks. In this thesis, we will describe Neural networks (NN), Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN) and Transformers.

2.3.1 Neural Networks

The simplest type of neural network is the feedforward neural network. It is a mathematical function that maps some input data to output values. The function is formed by combining many simpler functions.

The feed-forward operation is defined in 2.1. This is the output of a single neuron, or *perceptron* in the network where b is the bias, W is the trainable weights and $g()$ is the activation function. Stacking these over multiple layers – having the previous layer's output as input – is a *Multi-layer Perceptron* (MLP).

$$y = g(X^T W + b) \tag{2.1}$$

The activation function $g()$ gives the data non-linearity. The data is fitted along a line... One example of an activation function is the *Sigmoid* function. Defined in eq. 2.2 and plotted in Fig **??**, is a nonlinearity that fits a perceptron's output between 0 and 1. The function is often used in Autoencoders, RNNs, shallow CNNs, and MLPs.

$$g(a) = sigmoid(a) = \frac{1}{1 + e^{-a}} \tag{2.2}$$

Another example of an activation function is the rectified linear unit, or *ReLU*. Eq. 2.3 shows the simplicity of the function. if $a > 0$, the data is fitted linearly, if $a < 0$ the output will be 0. This function is commonly used as a non-linearity between convolutional layers.

$$g(a) = ReLU(a) = max(a, 0) \tag{2.3}$$

2.3.2 CNN - Convolutional Neural Networks

Convolutional neural networks are designed to handle data with a grid-like structure, such as a 1D grid (time series data) or image data in a 2D grid. The foundation for this NN comes from the mathematical operation of convolution, which is widely employed in signal and image processing. CNN is a simple MLP where matrix multiplications are replaced for convolutions between layers. Convolutions have three important aspects that improve the machine learning system. *Sparse interactions*, *parameter sharing* and *equivariant representations*. These networks also allow for input data of variable size.

Traditional NNs connect every output unit with every input unit. CNNs have

sparse interactions, reducing the kernel size lower than the input size. This reduces the number of parameters that need to be stored, reducing memory size and increasing statistical efficiency[8].

Parameter sharing allows for using the same parameters over different functions in the model. Traditional NNs have parameters w_i, which, from eq. 2.1, is defined as multiplication by one element of the input X_j and never revisited. If the parameters, or weights, were used over multiple inputs, this would be parameter sharing. The nature of the convolution, which convolves a filter window over a matrix, will allow the parameters to be used over different data locations[8].

Lastly, *equivariance*, means that if the input changes, the output changes linearly. This is useful when, for example, processing time series of data. A convolution produces a timeline that shows when different features appear in the input data. If we move an event later in time, the exact representation will appear in the output just later.

A common approach to a convolutional network layout is:

- A first layer performs convolutions over the input data. This gives a set of linear activations.
- A second layer which applies a nonlinear activation function to the linear activation. An example would be a ReLU. This is sometimes called the detector stage.
- Third layer, a pooling function is applied to further modify the layer's output.

Pooling functions replace the layer output at a certain location with a summary of statistics of the nearby outputs. Pooling assists with making the representations approximately invariant to small input translations. Invariance to local translation gives the model a better understanding of whether a property is present rather than the exact position. For example, if the objective is to classify faces in an image, pixel-perfect accuracy on the location of the eyes is not needed[8].

2.3.3 RNNs and Transformers

RNNs are neural networks that can process sequential information, such as time-series data or text. CNN models the spatial dependencies, whereas RNNs model the temporal dependencies in the data. In the previous section, we introduced parameter sharing. Recurrent networks also have parameter sharing, allowing them variable input data. Parameter sharing in this context is necessary to reduce redundancy in learning[8]. Analogously, the sequence "I bought a banana yesterday" and "Yesterday I bought a banana" have the same conceptual meaning, but the content is rearranged. With parameter sharing in RNNs, concepts do not need to be learned for every position in a sequence.

Efficient language models often require machine learning techniques specialising in sequential data, and RNNs have been widely adopted for this application. These language models consider their previous state (e.g. input data) when generating their current state. Vanilla RNNs proved incapable of learning long sequences. Nevertheless, vanilla RNNs demonstrated their limitations in effectively processing lengthy sequences. The issue arises from vanishing gradients within the backpropagation-through-time (BPTT) algorithm, causing vanilla RNNs to struggle when confronted with lengthy input sequences. This is primarily due to the RNN lacking access to crucial information about the sequence's initial stages as it reaches its end[9].

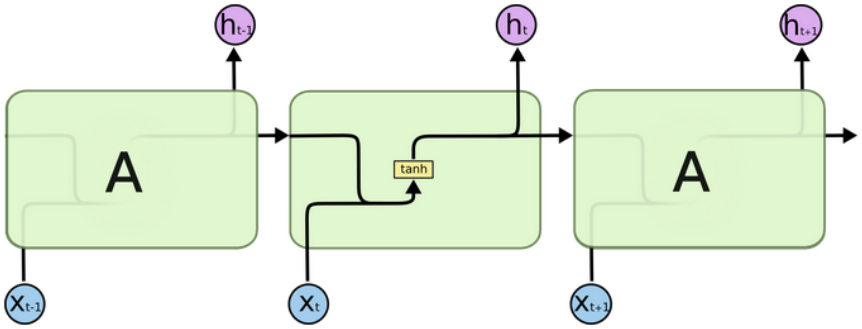

Figure 2.1: Figure of an vanilla cell, Figure from[1].

1

1. http://colah.github.io/posts/2015-08-Understanding-LSTMs/

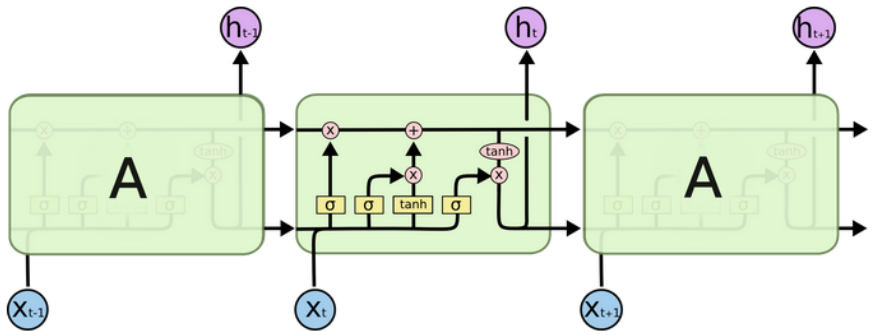

Figure 2.2: Figure of an LSTM cell, Figure from[1]

LSTM and GRU accounted for this by implementing information gates. These gates allow the model to decide which information to learn and which to forget. Fig. 2.1 shows the information flow in a regular NN cell, where the input data and the previous hidden state h_{t-1} are passed through a non-linearity which will be the output of the cell t hidden state h_t. The LSTM described in Fig. 2.2, on the other hand, has the ability to remove or add information to the cell state. Gates regulates the information. The gates are made by a sigmoid function which outputs a number between zero and one. If the sigmoid is high, or 1, the gate lets all information through, whereas the gate is 0, and it lets no information through. The LSTM cell is defined as in eq. 2.4.These give how the memory is kept and information is stored[10].

The first gate, the forget gate f_t, uses the previous hidden states h_{t-1} and the input data x_t to the cell to evaluate whether the information in the memory cell C_t should be kept or forgotten. The update gate i_t decides whether the memory cell should be updated with new information. The new information is defined as $\tilde{C}_t$. If the update gate i_t signals to completely update the information bank, all the information in candidate state $\tilde{C}_t$ is added to the new state C_t.

Furthermore, the cell's output o_t will be based on the cell state C_t. For example, if a language model just saw a subject, the cell might want to output information relevant to a verb. The cell's output is the information that it thinks is important.

$$\begin{aligned} f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\ i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\ \tilde{C}_t &= \sigma(W_C \cdot [h_{t-1}, x_t] + b_C) \\ C_t &= f_t \circledast C_{t-1} + i_t \circledast \tilde{C}_t \end{aligned} \tag{2.4}$$

The hidden state of a cell is defined in 2.6.

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \quad (2.5)$$
$$h_t = o_t \circledast tanh(C_t) \quad (2.6)$$

With the vanilla RNNs and LSTM, the model maintains a memory of previous information. The LSTMs allowed us to maintain a better information bank in the cell state. The sequential nature of recurrent neural networks precludes optimizations such as parallelizations during training, which is a critical factor in long sequence lengths.

2.3.4 Transformers

With the RNN's performance drawbacks becoming apparent, a new model architecture was introduced. The Transformer model eschewed the recurrency of the model and instead relies on an attention mechanism to draw even longer, global dependencies between input and output. This ubiquitous architecture introduced the self-attention mechanism. In contrast to previous sequence-to-sequence models, such as LSTMs, which process the input sequentially and rely on a fixed-length hidden state to encode the input, the transformers allow for variable sequence lengths.

The Transformer model embeds the input sequence to a higher-dimensional space, where the self-attention operates. The self-attention mechanism computes a weighted sum of the input sequence at each position. The similarities between the current and other sequence positions determine the weights. This allows the model to attend to different parts of the input sequence depending on the current context without relying on a fixed-length hidden state. [11]

Another mechanism in the architecture is the Multi-head attention. This allows the model to attend to multiple parts of the input sequence simultaneously and capture more complex relationships between different parts of the input sequence. Positional encoding of the input sequence was necessary to allow the model to use the order of the sequence. [11]

2.3.5 BERT - Bidirectional Encoder Representational Transformer

The BERT paper was introduced in 2018, with several innovations to Natural Language Processing (NLP) field. The paper introduced pre-training tasks, the

Masked Language Modelling and Next Sentence Prediction task [12] to the transformer architecture.

The **Masked language Modelling** task is done by masking some part of the input sequence and then predicting the masked input. The training of these tasks causes a mismatch between pre-training and fine-tuning since the masked token will not appear in the fine-tuning. Not always replacing masked sequence parts with the actual token mitigated this. The result of this task is to create a deep bidirectional model.

Furthermore, the **Next sentence Prediction** task is to make the language model capture relations between two sentences or sequences. This is done by selecting two sentences *A* and *B*, with a 50% probability of being the next sentence. [12]

2.4 ERS - Electronic Reporting System

The electronic reporting system used by the Norwegian Fisheries Directorate is called "Fangstregistrering" and is used by fishing vessels to report their catch and fishing activity to the Directorate. Multiple different messages are sent by vessels. Before a vessel departs, a *DEP* message is reported. This message reports which harbour the vessel are leaving, the time of department, the quantity of catch on deck at departure, and planned fishing activity. A vessel is required to send a *POR* message when the boat arrives. The message must be sent at most two hours before arrival to harbor[13].

A ***DCA*** message is required to be published at least one time daily and must be sent until an *POR* message is sent. The DCA message describes the position and time of when the fishing operation starts and ends. It also is required to write total catch, distributed over species in round weight (weight of the fish before it is gutted).

An example of the records in a DCA message can be seen in Table 1. The table shows one actual DCA message.

The Directorate of Fisheries has an open data policy, meaning they publish all data from the reporting system for fishing vessels larger than 15 meters. Currently, the publishing is done by periodically publishing CSV files on their webpages [14].

3

Related work

Multitudes of decision support systems for marine ecology exist. Understanding the "state of the art" is essential for technological advancements. For this thesis, related machine learning- and survey- systems are presented in this section as experiences and development of theories can be built upon.

3.1 EchoBERT

We have previously introduced the Transformers and the improvements made with *BERT*. **EchoBERT**[9] is a Transformer-based approach for echograms. The model was created for behaviour detecting and monitoring fish cage populations. The paper presents an application of sequence models in echogram behaviour classification and a novel pre-training task adapted to the hydroacoustic domain.

As presented in section **??** the echograms that are output from echosounders visualize time along the x-axis and depth on the y-axis. The echograms have spatial and temporal information and can be seen as many time series for various depth levels stacked on top of each other.

The paper presents the novelty of using sequence models for echo data. Language modelling tasks have fixed vocabulary in the corpus that is trained on. The input sequences for echo are continuous data. This requires the model to

use raw echo vectors instead of word embedding as previous work did.

The paper introduces a novel task for echograms. **Next Time Slice Prediction** is a task for the model to understand long-term dependencies in the echograms. A fish's school position in a one-time slice affects future school positions. Furthermore, with inspiration from BERT, a substituted Vector prediction task is used to train the bidirectional transformer model, where the original '[MASK]' token is substituted from a random vector sampled from the training dataset. The task for the model is to predict whether the vector is the true vector or substituted. The result of this task gives the model a bidirectional aspect without the mismatch of pre-training and fine-tuning.

The model's classification task was to detect diseases in fish cages. With this architecture, the model outperformed LSTM with a substantial margin on the MCC scoring.

3.2 Machine learning pipelines

The novelty of the proposed approach requires us to adapt existing theories and artifacts to accommodate our goal. Various methods have been used for species classification with the use of machine learning, and this thesis will extend their ideas and shift the constraints to facilitate new artifacts for our method.

3.2.1 CRIMAC

CRIMAC[1] is the Centre for Research-based Innovation in Marine Acoustic Abundance Estimation and Species Classification and is a research initiative aiming to develop and improve methods for stock assessments using acoustic data.

The SFI has contributed a pipeline for pre-processing, detection of the seabed and various machine learning architectures for hydroacoustics. The data that has been used in their most recent works are from an EK60 echosounder, which operated at $[18, 38, 129, 200]$ kHz. This data was collected from The Norwegian Institute of Marine Research on their annual trawl survey of sandeel areas.

1. The centres' webpage `https://www.crimac.no`

Pre-processing

Papers [15], [2] introduce the methods used for pre-processing the data of the sand eel survey. The data must be interpolated to a common time-range grid across the frequencies. The product of this step is a tensor of size $[4, N_p, N_r]$ where N_p is the number of pings (time, or x-axis in an echogram) and N_r is the range (y-axis in an echogram). The primary frequency of each sample is set to 200 kHz, considering the sand eels signal-to-noise ratio [16], meaning that this main frequency aligns with every other frequency data.

Data preparation for machine learning libraries is an important part of the process. They compose dataset samples with the xarrays [17], allowing for N-dimensional labelled memory maps with out-of-core computation for large datasets which does not fit memory.

Bottom detection

Detection of seabed in data is vital for echo integration. Uncalibrated echo data from echo sounders often experience noise from external noises, and echograms have garbage intensities below the seabed. This makes it critical to have techniques for the detection of the seabed.

The project team has implemented multiple echo processing techniques for bottom classification [18]. Three of them are used separately or in conjunction with each other. These are:

The **Simple** algorithm is a fast algorithm which uses the maximum back-scatter intensity in each ping to find a bottom depth
Secondly, the **Angles** algorithm uses the spilt beam angles to detect the bottom depth. This is used when the data contains a slopy bottom, and one or both split beam angles vary linearly with depth.

The **Edge** algorithm convolves over the back-scatter s_v to find bottom candidates. The candidate is selected based on the highest quality, meaning the width and prominence of the convolution peaks. The three aforementioned algorithms are used in a **combined** algorithm, which first uses the simple algorithm, for then to define bottom candidates with angles, if the angle regression fits succinctly otherwise, use the edge algorithm [19].

Classification

Recent works include different machine learning models applied to the sandeel data. Current methods rely on much annotated training data, which is only acquired by manual annotation processes. Thus, semisupervised learning has been considered in [1] and [2].

The preceding publications present a novel method for leveraging small amounts of annotated data samples with vast amounts of unannotated data samples. The model has a *clustering* and a *classification* objective. The first objective is to exploit the underlying structures of all data in the corpus. Figure 3.1 gives an overview of the proposed model's operation. Inference takes patches of echograms for all frequencies included in the data. And classifies it as Sandeel, other species or background.

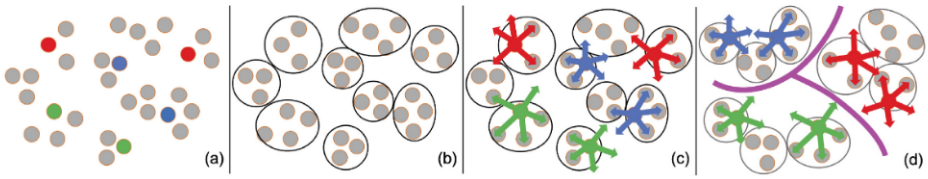

Figure 3. Overview of the proposed method. Each point represents the extracted patch, where the point in gray is unannotated while the points in color (red, green, or blue) indicates the annotated one with respect to the class. (a) The training data occupy an arbitrary space. (b) The clustering objective helps to form clusters regardless of the annotation. (c) The available annotated data and the classification objective optimize the CNN in a supervised manner. (d) The iteration of (b) and (c) constructs the decision boundary with respect to given classes, where the unannotated points take their place inside the boundary according to their own clusters.

Figure 3.1: Illustration of the method for [2], taken from the original publication, licensed[2]

2

The latter of the works [1] utilises the same assumption with a semisupervised approach. Still, instead of classifying the patches, the classification objective is to utilise semantic segmentation to obtain a pixel-level classification task. Inspired by U-Net, [20], and previous work of [15] on the echosounder data, an encoder-decoder architecture with skip connections is postulated.

2. https://creativecommons.org/licenses/by/4.0/

4

Design

In this section, we will describe the requirements and design for the pipeline.

The pipeline consists of multiple components with different requirements. Figure 4.1 gives a high-level architecture of the design and components of a DSS for fisheries. This thesis delimits itself to the preprocessing and classification components, as enclosed in red marking and text in the Figure. All components are created with modularity in mind, and each component has encapsulated its functional requirements to fulfil this.

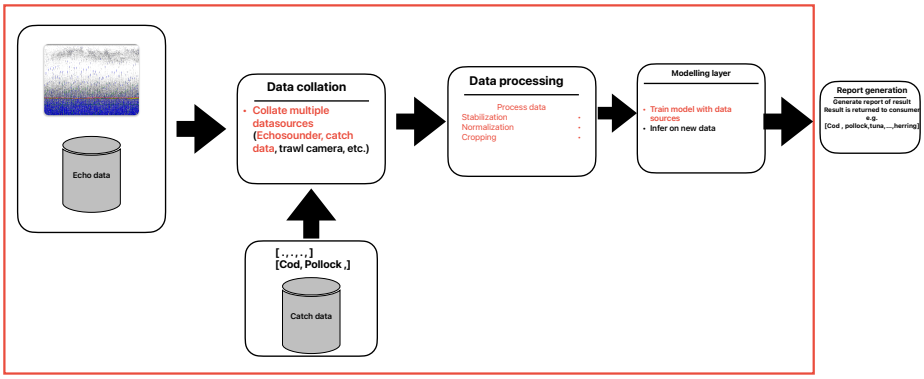

Figure 4.1: Illustration showing the design of a pipeline for a decision support system using echo data

4.1 Requirements

The requirements of the thesis derive from findings in the literature, as well as the context from section 1.5. In this section, we outline the requirements for the system. Henceforth, each subsection describes the requirements in detail.

- Must be able to process echosounder data (ek60 , ek80) from Kongsberg's .raw files
- The read echo examples must be *stabilized* and unwanted features removed
- The echo examples should be annotated with catch data
- Computational overhead must be taken into account when developing
- Must support processing for training and inference. Before training, a dataset should be generated from a set of echo data.
- Data loading into the deep learning model should be done efficiently.

4.1.1 Echo data processing

The raw data file from Kongsberg is a proprietary file format – *.raw* – composed of XML or binary datagrams[1]. The file contains data collected from Kongsbergs EK80/EK60 echosounders. These files must be processed into a simpler, more interpretable form of data structure to easily adapt to new tasks.

4.1.2 Echo data stabilization

In this thesis, we assume imperfections in the echo data. Noise and poor calibration are common in In-situ operations. Random noise needs to be filtered out,

1. https://www.simrad.online/ek80/interface/ek80_interface_en_a4.pdf#GUID-09211E50-E0C2-4495-B12F-FD915FC1F472

and seabed detection is required for cropping. Unwanted and dead features must be removed to remove potential data discrepancies.

4.1.3 Echo annotation

To utilize supervised learning, we either need a large annotated feature space or we need to use other means of modelling. Manual annotation methods are infeasible, so echo examples must be annotated with catch data. In section 2.4, we explored the catch messages *DCA*, containing multiple tuples of catches. The echo data should be labelled with round weight, species and date.

Moreover, it is crucial to consider *computational overhead* and efficiency in the context of collating the echo data and catching messages. This process involves combining two large data sources and performing computations on them simultaneously. Therefore, it is essential to address parallelization and effective resource usage complexities. Efficient utilization of computational resources and parallelization techniques should be considered to ensure optimal processing performance and minimize computational overhead. This consideration will contribute to the overall efficiency and effectiveness of the system.

4.1.4 Model Integration

To effectively utilise the echo examples, supporting their usage in both the training and inference phases is essential. Before training, a dataset must be generated by processing the raw files. This dataset will serve as the input for training the deep learning model. The model integration should ensure seamless integration of the processed echo examples into the training pipeline, enabling efficient and effective model training. Additionally, the trained model should be capable of performing inference tasks, providing predictions and insights based on new input data.

4.1.5 Data loader

Efficient data loading is a crucial requirement to facilitate the smooth processing of batches for both training and inference tasks. The data loader component should be designed to load and process the data in batches efficiently. This involves considerations such as parallelization, effective memory management, and data preprocessing techniques to ensure that the input data is readily available for training or inference.

4.2 Proposed Design

The proposed design of the pipeline involves dividing the requirements into different layers. The thesis primarily focuses on the *processing* and the *model* layer.

4.2.1 Processing Layer

To fulfil the requirements of echo processing, stabilization and annotation, we propose the processing layer with three components. The *Processor*, *Fetcher*, and *Collator*.

Firstly, the **Processor** component is responsible for reading .raw files and processing them to a more interpretable and computational format. The echo data, or s_v, is combined with time and depth and parsed NMEA data.

The second component, **Fetcher**, is responsible as an adapter between the data source containing the catch data and the aforementioned components, Processor and Collator. The Fetcher reads and stores *DCA* messages and selects the relevant tuples that need to fulfil the collation task.

The primary requirement of the **Collator** component is to collate two data sources based on positional attributes. It creates a species distribution mapped to an echo transect, providing information about its biota based on catch data.

To train a model, a dataset needs to be created. The pipeline supports processing a set of echo data to generate this dataset. The dataset is generated by the three components and stored on disk as a representative echo example per file and a serializable object containing annotation-specific data.

An adapter is created for interchangeable data streams. Upon creation of a dataset, all pre-processed features are stored on disk. Inference on the pipeline is made by pre-processing (unknown) data and sent to the classification layer with data streams**??**.

4.3 Proposed model architecture and requirements

This section outlines the proposed design of the model architecture and its training process. The model architecture will be designed specifically for the task of modelling fish abundance using hydroacoustic data. It should be capable of processing the processed echo data as input and predicting an estimate of fish abundance based on the collated catch data annotations.

Related work has described models adapted to the echo domain. With CRIMACs models of semi-supervised learning, both encoder-decoder networks, and VGG-16, showed significance in classifying sandeel by creating pseudo-labels with clustering. Other works; include EchoBERT, which employs NLP-inspired BERT, where a spatiotemporal assumption is made. As the novelty in the thesis does not rely on the model architecture, details about the architecture are referred to in the respecting works [1, 2, 9]. The thesis employed the EchoBERT model as its base architecture and extended it for our requirements.

The first requirement for the model was to be able to do both *regression* task and *classification* tasks. As the annotation method is one of the main contributions, we wanted to model the data for both tasks.

Secondly, to facilitate the number of species in the model, the model's output layer needs to change according to the label set or manually selected variably. The label set is not a static amount of the number of species. The collation criterion is based on thresholds, potentially including more catch messages with unique species. This required the model to interchange its output dimension based on the input training data.

Finally, the loss function needs to be able to model temporal proximity in time. If the echo data date is closer in time, the backpropagation should represent that. The biota in a location is based on many parameters, one of them being time. By considering the temporal proximity in the loss function, the model can learn to capture the changes in fish abundance over time, potentially leading to better predictions.

4.3.1 Objective function

The proposed model will have interchangeable criteria for the objective function to incorporate the ability to perform both regression and classification tasks. This means that during training, the model can be configured to optimize for regression (predicting continuous fish abundance values) or classification (pre-

dicting discrete labels for fish abundance ranges or species presence/absence). The loss functions used for the tasks are described in section 4.3.2.

Additionally, the output layer of the model will be designed to variably change based on the label set or manually selected species. This flexibility allows the model to adapt to different scenarios where the number of target fish species can vary. The model will dynamically adjust its output dimension based on the input training data, ensuring compatibility with different label sets.

4.3.2 Loss function with temporal proximity weighting

To implement the requirement of weighing loss functions based on catch date, or when the transect time t_{tr} is sampled, labels found from the data processing layer will contain times $t_L = t1, t2, ..., t_n$. Thus, the model must allow times in t_L where $t_{tr} \approx t_L$ is more relevant to the loss output. Past and future labels wrt. Transect time must be weighed as lower-grade information compared to the present labels.

As the task is to predict an output or a vector of outputs, mean squared error is employed. *MSE* is defined as eq. 4.1

$$MSE = \frac{1}{m}\sum_i (\hat{y} - y)_i^2 = \frac{1}{m}||\hat{y} - y||_2^2 \tag{4.1}$$

where $\hat{y}$ is the predicted output, and y is the ground truth label. The aforementioned loss function is used for regression tasks. To make it possible for the model to classify species instead, binary cross-entropy, defined as

$$BCE = -(y \log(p) + (1 - y) \log(1 - p)) \tag{4.2}$$

where p is the predicted probability, and y is the true label (0 or 1).

Lower grade information is calculated by finding time differences between t_{tr} and t_L and fitting the information scores along a bell curve. If the time proximity is close, the backpropagation should take advantage of the loss of the node. As shown in Figure 4.2, dates are aligned along a Gaussian distribution. The x-axis represents days, and the y-axis represents the output value for a weight w.

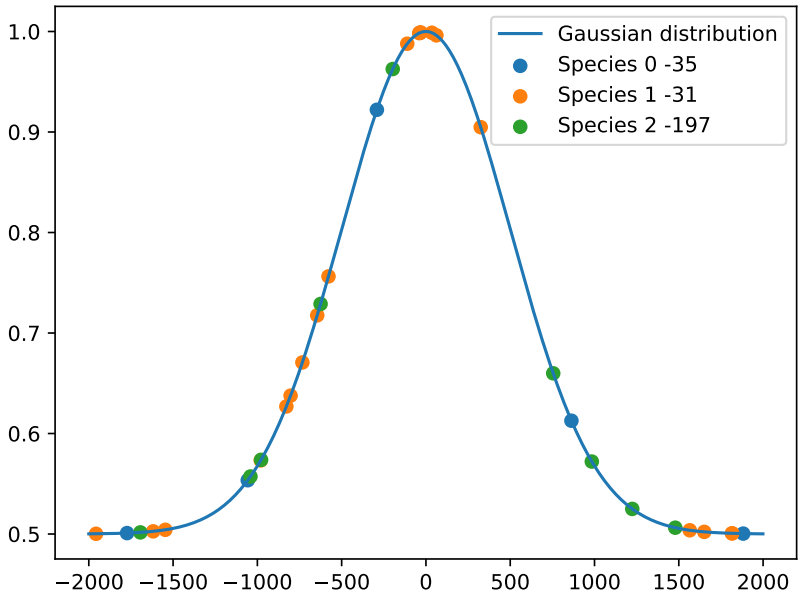

Figure 4.2: Graphic of time deltas fitted along a bell curve for illustration

Overall, the proposed model architecture will build upon the strengths of the EchoBERT model while introducing modifications to address the specific requirements of modelling fish abundance using hydroacoustic data. By incorporating interchangeable criteria, flexible output layers, and accounting for temporal proximity in the loss function, the model aims to provide accurate predictions and adaptability to varying scenarios and label sets.

5 Implementation

In this chapter, we present the implementation of the pipeline. Because each component's specific requirements and each layer in the pipeline must be presented individually, we will propose methods explaining how the design is implemented in our thesis and cover implementation-specific details.

5.1 Programming language

To enable an iterative approach and fast evaluation of ideas, Python[21] was chosen as the language for all components. The language enables the usage of widely used packages in the domain for the thesis, xarray[17], which enables for processing of labelled data arrays in parallel. Moreover, PyTorch is a high-performance deep-learning library widely used. Both related works models were developed in this framework, using them in the thesis became natural.

Lastly, to facilitate the requirement of fast processing of echo and catch data, *Numba*[22] was used to create *Embarassingly parallel* methods.

5.2 Data processing layer

5.2.1 Echo data processing

The echo-sounders .raw files are processed with **pyEcholab**[23], an open-source package in Python enabling the processing of Simrad/Kongsberg data files. With pyEcholab, you can read the .raw files from the echo-sounder and extract the required information. One of the important calculations performed during the reading process is the estimation of the volumetric backscatter coefficient, denoted as s_v.

After calculating the s_v values, the data is stored in an *xarray* dataset, a powerful data structure in Python for handling multi-dimensional arrays with labelled axes. Along with the s_v values, the dataset also incorporates auxiliary information obtained from the transect. This auxiliary information includes parameters derived from the parsed NMEA data, such as latitude, longitude, heave (vertical motion of the vessel), pitch (rotation around the lateral axis of the vessel), roll (rotation around the longitudinal axis of the vessel), and transducer angles.

Organizing the echo data and auxiliary information into an xarray dataset makes it easier to manage and analyze the data in a structured manner. The dataset allows for efficient retrieval and processing of the hydroacoustic data, enabling subsequent collation with the catch data as described earlier.

5.2.2 Echo data stabilization

We designed methods of filtering and seabed cropping from the requirements to reduce noise and enhance the data quality. One of the common sources of noise in the echogram is random fluctuations, as depicted in Figure **??**.

Noise filtering

To mitigate this noise, we employed spatial filters. A Spatial-temporal filter was implemented as the noise appeared sporadic and noticeable periodicity was exhibited. Spatial-ordering filters efficiently remove random noise without requiring computationally expensive operations such as Fourier Transforms. This approach is beneficial for processing large quantities of data efficiently and assists in the optimization of the performance of the processing layer.

The implementation of the spatial-ordered filter, or median filter, is shown in 1. The implementation is parallelized with Numba.

```python
@nb.njit(parallel=True)
def median_filter(x,size=3):
    """
    Median filter
    """
    x_new = np.zeros_like(x)
    for i in nb.prange(x.shape[0]):
        for j in nb.prange(x.shape[1]):
            if i == 0 or j == 0 or i == x.shape[0] - 1 or j == x.shape[1] - 1:
                x_new[i,j] = x[i,j]
                continue
            x_new[i,j] = np.median(x[i-size:i+size,j-size:j+size])

    return x_new
```

Listing 1: Median filter

Seabed Cropping

The requirement of echo stabilization 4.1.2 also describes the need for seabed cropping. Various image processing techniques can determine the indexes of the seabed. Section 3.2.1 we described CRIMACs bottom detection algorithms and the *simple* algorithm has been one of the methods employed in the thesis. Sometimes, echosounder data includes *.bot* files which is a complementary file containing information on the bottom ranges. These files are often included with calibrated data and give better results than approximation methods.

Method for finding the index in the echogram where the seabed lies for the corresponding ping is created and listed in Code listing 2. This method employs the .bot data file.

Figure 5.1 shows the s_v before and after cropping. Figure 5.2 shows s_v before and after bottom fill.

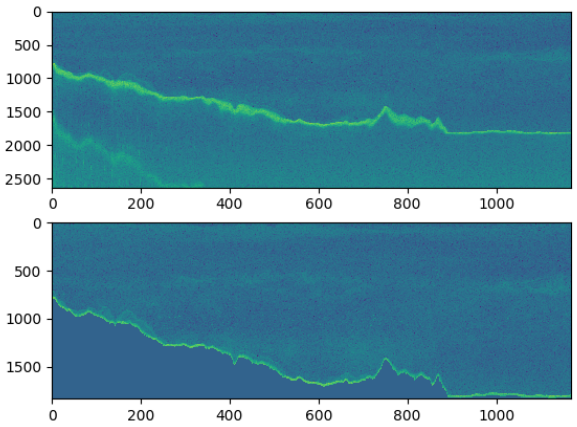

Figure 5.1: Cropped image on seabed
(Top) Original seabed image (Bottom) cropped and flood-filled to the seabed

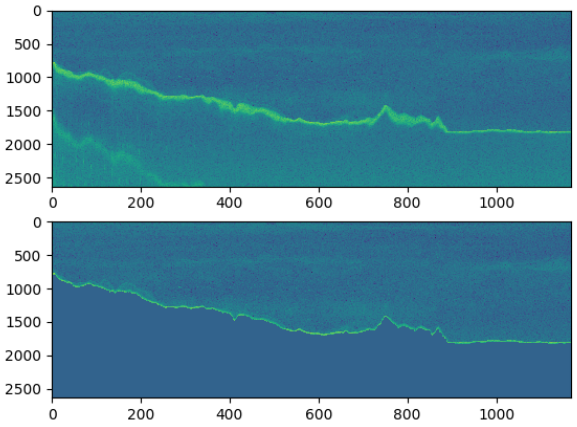

Figure 5.2: Flood filled to seabed
(Top) Original seabed image, (Bottom) flood-filled to the seabed

```python
def crop_matrix_bottom(ds,crop=0):
    """
    Crop matrix from bottom line
    Finds largest index, and masks out the rest
    """
    OFFSET = 2

    if ds is None:
        return

    bottom_data = ds['bottom'].data[0]
    range = ds['range'].data

    largest_bottom_data = np.max(bottom_data)
    index = np.argmax(range > largest_bottom_data )

    x = ds

    if crop:
        x = ds.isel(range=slice(0,index + OFFSET))

    for i, bottom in enumerate(bottom_data):
        if i == bottom_data.shape[0] - 1:
            continue
        try:
            biggest_index = np.argmax(x.range.data > bottom)
        except IndexError:
            continue

        x.sv[0,i,biggest_index:] = -90

    return x

```

Listing 2: Example cropping

5.2.3 Echo annotation

The echo annotation methods are implemented using two main components.

Firstly, the catch data is stored in a PostgreSQL instance. The fetcher component

is responsible for streaming all the records from the database into a *DataArray* from the **xarray** library.

Since the catch data is periodically updated with new records from the Directorate of Fisheries, the existing records are assumed to remain static. All multi-dimensional data objects created from the database are written to disk to enable efficient retrieval and processing of the catch data. This allows for quick access to the catch data during processing.

After the retrieval of the catch data, the processing component performs the collation of the echo data with the catch data object.

5.2.4 Collation criterion

Both the hydroacoustic echo data and the catch data contain positional information. The method used for collation is indexing the catch messages based on the distance between the hydroacoustic ping and the catch locations. This distance calculation is accomplished using the Haversine formula.

The haversine formula is a mathematical equation for calculating the distance between two points on the surface of a sphere. This is useful in navigation and GIS for estimating the distance between two points by their latitude and longitude.

$$d = 2R \arcsin \sqrt{\sin^2 \frac{\Delta\phi}{2} + \cos \phi_1 \cos \phi_2 \sin^2 \frac{\Delta\lambda}{2}} \tag{5.1}$$

Equation 5.1 calculates the great-circle distance d between two points, with R being the Earth's radius, ϕ_1 and ϕ_2 the latitudes of two points (radians), $\Delta\phi$ the difference in latitudes $\phi_2 - \phi_2$ and $\Delta\lambda$ is the difference in longitude between the two points. The output d (distance) unit is given in m or km. This algorithm determines the distance in a given radius of the hydro-acoustic measurements, as provided in Figure 5.3.

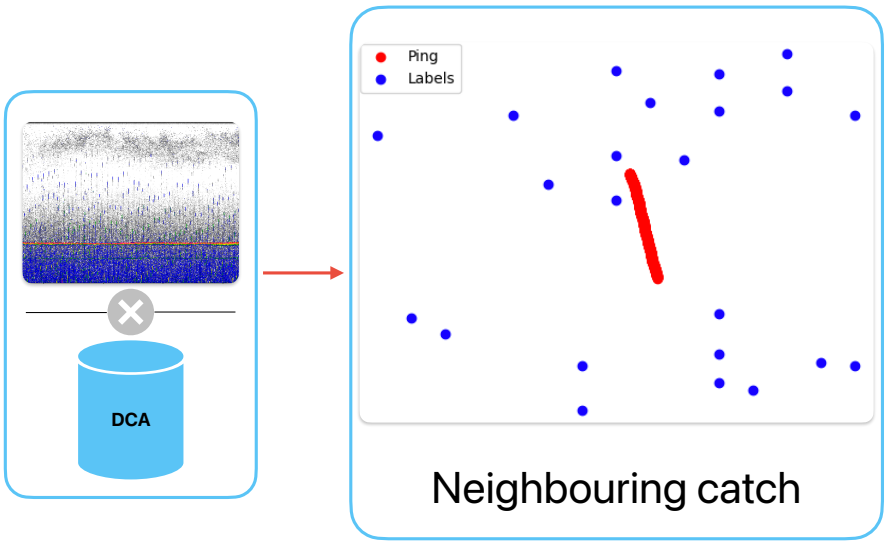

Figure 5.3: Illustration of collation of hydroacoustics ping coordinates to DCA message positions
(right) illustrates in red, the positional data from the hydroacoustics and the blue points are the neighbouring catch messages within a $1km$ radius of the hydroacoustics coordinates.

In Figure 5.3, the points in blue are the neighbouring catch messages within a $1km$ radius of the hydroacoustics coordinates (red).

Haversine calculation

To allow for fast inference and real-time predictions on data, one of the requirements that the annotation method needed was to have a reasonable compute overhead. The code listing 3 shows the implementation of eq. (5.1).

```

@nb.njit(fastmath=True)
def calculate_haversine(lat_transect,lat_labels,lon_transect,lon_labels):
    lon_transect,lat_transect = np.radians(lon_transect),np.radians(lat_transect)
    lon_labels,lat_labels = np.radians(lon_labels),np.radians(lat_labels)

    dlon = lon_labels - lon_transect
    dlat = lat_labels - lat_transect

    a = np.sin(dlat/2.0)**2 + np.cos(lat_transect) * \
            np.cos(lat_labels) * np.sin(dlon/2.0)**2

    c = 2 * np.arcsin(np.sqrt(a))

    return 6367 * c

```

Listing 3: Haversine computation

As the Haversine method is applied for each index $i \in A$ where A is vectors with latitude and longitudinal positions of the transect, denoted as A_{lat} and A_{lon} of size N. There is also $j \in B$, corresponding to the positional information from the catch reports, denoted as B_{lat} and B_{lon} of size M. The resulting distance matrix D, of size N, M reflects all Haversine distances, based on every index $i, j \in A, B$.

The code listing 4 shows the implementation of this functionality. Along with the distance matrix, indices are calculated based on a kilometre threshold. This enables us to index our Dataset based on the distances within our sampling region.

```
1
2  @nb.njit(fastmath=True,parallel=True)
3  def calculate_haversine_unvectorized(lats_transect,lats_labels,\
4          lons_transect,lons_labels,threshold=10.):
5
6      lat_lon_tr = np.vstack((lats_transect,lons_transect))
7      lat_lon_labels = np.vstack((lats_labels,lons_labels))
8
9      array = np.zeros((lats_transect.shape[0],lons_labels.shape[0]))
10
11     for i in prange(array.shape[0]):
12         lat_i,lon_i = lats_transect[i],lons_transect[i]
13         for j in prange(array.shape[1]):
14             lat_j,lon_j = lats_labels[j],lons_labels[j]
15             km = calculate_haversine(lat_i,lat_j,lon_i,lon_j)
16
17             array[i][j] = km
18
19     indexes = np.argwhere(array < threshold)
20
21     return array, indexes
22
```

Listing 4: Distance matrix computation

The method in code listing 4 is unvectorized and uses the *embarrassingly parallel* functionality of *Numba*[22] to allow for significant speedup compared to other methods we implemented.

Label selection

The labels are selected based on the kilometre threshold taken as a parameter in code listing 4. Recall the first vector B, which is the positional information of the annotated data. By finding the indices in vector B, it follows that the indices represent a Message ID in the *DCA*. Thus, it has a round weight, date and species code associated with it.

All message IDs in that area are also found by finding all the unique indices in the M dimension found from the distance matrix D. The code listing 5 is the method for finding all unique indices in the catch data, and the code listing 6

shows the rest of the grouping and label collation.

The found labels represent the weight of each species found in the transect. As seen from the example output label in listing 7, the species codes *[GUG ,MAC ,SAN , WHG, HER and HAD]* is found, with corresponding summed weight and dates.

```
def convert_to_unique_indexes(indices,axis=0): # from utils.py
    """
    Convert indices to unique indexes
    @input : np.array(2,X)
    @returns : indices for specified datasource
    """
    return np.unique(indices[:,axis])


class Collator: # snippet of collator class in collator.py
    ...

    def collate(self,ds,fname,plot=False):

        labels_lat, labels_lon = np.array(self.labels['Startposisjon bredde'].data),\
                                 np.array(self.labels['Startposisjon lengde'].data)

        lat_transect = np.array(ds.lat.data[0])
        lon_transect = np.array(ds.lon.data[0])

        ...

        distance_matrix,indices = calculate_haversine_unvectorized(lat_transect,labels_lat,\
            lon_transect,labels_lon,threshold=DISTANCE_KM_THRESHOLD)

        indices = convert_to_unique_indexes(indices,axis=1)

```

Listing 5: Unique indices

```
-1
        selected_labels = self.labels.isel(dim_0=indices)

        selected_labels = selected_labels.dropna(dim='dim_0',how='any')

        try:
            selected_labels_grouped = selected_labels.groupby('Melding ID')
        except Exception:
            return {}

        groups = selected_labels_grouped.groups

        dict = {}

        for group in groups:
            group_labels = selected_labels_grouped[group]
            for group_art_key, group_art_ds in list(group_labels.groupby("Art FAO (kode)")):
                if group_art_key not in dict:
                    dict[group_art_key] = {'weight':[],'date':[]}

                largest_version = group_art_ds.isel(dim_0=-1)

                dict[group_art_key]['weight'].append(largest_version["Rundvekt"].data)
                dict[group_art_key]['date'].append(str(largest_version["Startdato"].data))

        for art in dict:
            dict[art]['weight'] = np.sum(dict[art]['weight'])
            dict[art]["date"] = list(np.unique(dict[art]["date"]))

        return dict


-
```

Listing 6: Index grouping and storage

```
{
    "GUG": {
        "weight": 312.0,
        "date": [
            "03.05.2021",
            "04.05.2021",
            "06.05.2018",
            "11.05.2018"
        ]
    },
    "MAC": {
        "weight": 7635.0,
        "date": [
            "03.05.2021",
            "04.05.2021",
            "06.05.2018",
            "11.05.2018",
            "13.05.2018",
            "18.05.2018"
        ]
    },
    "SAN": {
        "weight": 1098000.0,
        "date": [
            "03.05.2021",
            "04.05.2021",
            "06.05.2018",
            "11.05.2018",
            "12.05.2018",
            "13.05.2018",
            "16.05.2018",
            "18.05.2018"
        ]
    },
    "WHG": {
        "weight": 400.0,
        "date": [
            "03.05.2021",
            "06.05.2018",
            "11.05.2018"
        ]
    },
    "HER": {
        "weight": 5200.0,
        "date": [
            "13.05.2018",
            "18.05.2018"
        ]
    },

}

```

5.3 Model layer

This section describes the proposed model's implementation and the requirements devised in section 4.3.

5.3.1 Model integration and Data loading

The dataset is created using the *data processing layer*, and store the response examples on disk. Each example is stored in a unified folder, where each example is a *.zarr* formated multi-dimensional xarray datasets. We processed examples in stages, meaning the processing of *sv* and auxiliary data is processed, stored, and afterwards processed in the collator.

Pytorch comes with a *dataloader* module, allowing developers to create dataset classes which fetch examples and annotations. This component is a parallelizable task. To be able to use this component, we required storage of all echo examples in a different data structure and file format than previously created. Thus, all s_v data was converted into numpy arrays and stored on disk in numpy format.

The last requirement for integrating the data source into the model was that the s_v data was segmented into patches of equally large shapes. This was done by slicing each echo example into patches. Both the spatial and temporal resolution of the echo data was segmented. By splitting an example of size NxM in K parts, returns an array of $KxNx(M/s)$ echo examples, where s is the size divided to. The code can be seen in listing 8. If the shape of the example is not divisible by s, the last array split returns a non-satisfactory shape and is discarded. The corresponding segments are stored on disk and are ready for the data loader.

```
def segment_image(sv,segment_size=512):
    """
    Create patches of size segment_size from sv
    """

    return np.array_split(sv,sv.shape[1]// segment_size,axis=1)


def segment_dir(dir):
    import os

    files = os.listdir(dir)

    for file in files:
        sv = segment_image(load_npy(dir+file))
        for i in range(len(sv)-1):
            discard_last = sv[i].shape[1] < 512
            if discard_last:
                continue
            segment = sv2[i][:,:512,:]
```

Listing 8: segmentation of echo examples

Echo data loader

The model's data loader is created using PyTorch's DataLoader component. The programmer defines a class describing how the examples are loaded, along with the annotations. The DataLoader is responsible for efficiently loading data from disk and applying any necessary processing, such as transformations and augmentation, to both the input data and the corresponding labels.

Each example retrieved by the data loader is transformed. We will first announce the methods applied to each echo example and, afterwards, the transformations of the targets.

The echo data is normalized and *arranged*. The normalization step is done as presented in Code listing 11. The code scales the input image by subtracting the mean and dividing it by the standard deviation and max value. The arrangement of the data is implemented by the related work EchoBERT[9] even though the paper implemented the method for a continuous echo of the cages, the same principle is implemented in this thesis.

Each echo example has corresponding target labels. We processed four different labels, each evaluated by a different threshold on the collation criterion. We employ the threshold as a hyperparameter in the model. The corresponding label for the selected threshold is read and transformed into the desired label.

The labels are originally represented as presented in listing 7. We transform the labels into a stochastic distribution for the regression task or a multi-hot encoded array for classification.

The Stochastic distribution is calculated by

$$y_i = \frac{y_i}{sum(y)} \tag{5.2}$$

where y is the target vector and i is an target class. means that each target class is divided by the sum of the target vector.

The shape of the label vector is dependent on the desired task. We proposed a method for either applying a vector representing all species in the label set or specification of desired species as output vector. For example, the labels can represent one species, sand eel ("SAN"), corresponding to a binary classification or a regression task with one output. This is implemented as shown in code listing 9.

```python
def transform_labels_json(annotation : dict,truth:str,selection : list = None,onehot : int = 1):
    '''
    Param:
    Annotation: the examples label.
    truth: representing the date echo data is recorded.
    selection: list of FAO codes that should be learnt
    onehot: bool to determine the representation
    '''

    sz = len(selection)

    arr = np.zeros(sz)

    date_arr = []

    for key in selection:
        label = annotation.get(key,None)
        if label :
            arr[selection.index(key)] = 1 if onehot else label['weight']
            del annotation[key]
            date_arr.append(create_delta_time(truth, label['date']))
        else: # if no date, still insert empty list to ascertain right shape.
            date_arr.append([]) # no date for this label

    if not onehot and np.sum(arr) > 0:
        arr = np.clip(arr,1e-3,np.max(arr))
        arr = arr / np.sum(arr)
        arr = np.clip(arr,1e-3,1 - 1e-3)

    return arr, date_arr,selection

```

Listing 9: Transformation method for labels

As said, the number of output neurons depends on the use case. We have implemented methods for the multi-vector case that find all unique species codes for a corresponding threshold. This gives the model a label set corresponding to all existing species codes for the threshold. The implementation is enclosed in code listing 10.

```python
def find_max_number_species_code(dir:str = 'ds/labels_crimac_2021/', T="_5"):
    max_species = 0
    existing_label = np.zeros(len(os.listdir(dir)))
    selection = set([])

    for i,file in enumerate(os.listdir(dir)):
        if file.endswith(T+".json"):
            ds = load_json(dir+file)
            max_species = max(max_species,len(ds.keys()))

            if len(ds.keys()) > 0:
                existing_label[i] = 1
                keys = set([*ds])
                selection = selection.union(keys)

    selection = list(selection)
    selection.sort()

    return len(selection),selection
```

Listing 10: Find unique species in label set

The last objective of the target transformation is to create the data objects for the temporal proximity weights. The implementation is discussed in section 5.3.3.

```python
def normalize_sv(sv):
    mean = torch.mean(sv)
    std = torch.std(sv)

    sv = (sv- mean)/std

    sv = sv / torch.max(sv)
    return sv
```

Listing 11: Normalize echo data

A synthetic data class, originally from EchoBERT, is modified for the purpose of this thesis. This class is used for the pre-training tasks described in related

work, section 3.1. The modifications done compared to the original data loader are the method of retrieving examples from disk, the dimensionality of the data and the transformations as presented previously.

5.3.2 Model implementation

The model layer was implemented using PyTorch and PyTorch Lightning[24]. PyTorch Lightning is a wrapper for PyTorch enabling researchers to simplify the training and implementation of deep learning models. With abstractions and high-level interfaces, the researchers and practitioners can focus on the model's logic rather than boilerplate code.

The implementation was created by defining a Lightning class and its required methods. The model is, as said, a modified version of EchoBERT.

The Lightning module requires you to implement these methods for a working framework:

- **Forward**: Feedforward logic of model
- **Configure optimizers**: Configures the optimizers used in the model
- **Step functions**: Step functions for train, validation test step.

The first method, *Forward*, is the logic of the output neurons. Inspired by EchoBERT, the logic is divided between the pretraining and finetuning tasks. The pretraining task is retrieved from the implementation of EchoBERT[1]. But the classification task is adapted for the thesis use case. See code listing 12 of the forward implementation for reference. See Appendix **??** for the full model implementation.

The second method configures the optimizers used. As the model architecture is based on EchoBERT[9], we chose to employ the same optimizers, *AdamW*[25] and OneCycleLR[26].

The step functions implemented for training, validation and test steps use the forward method mentioned earlier.

1. Repository link: `https://github.com/haakom/EchoBERT`

```python
class LitModel(pl.LightningModule):

    # ....
    def forward(self, batch,batch_idx,forward_type,training=True):
        src = batch["enc"]
        dec = batch["dec"]

        if self.classification:
            if self.temporal:
                weights = batch["date"]
                weights = calculate_loss_resampling_weight(weights,sig=self.sig)
                weights = torch.tensor(weights)
            else:
                weights = torch.ones_like(batch["target"])

            weights = weights.to(self.device)

            preds = self.model(src.float(),dec.float(),None,None)
            if self.criterion == "mse":
                loss = F.mse_loss(preds,targets,reduction="none")
                loss = (loss*weights).sum() / loss.sum()

                r2 = r2_score(targets.detach().cpu(),preds.detach().cpu())
                mae = self.mae(preds,targets).to(self.device)
                return loss,[],_,preds
            else:
                loss = F.binary_cross_entropy_with_logits(preds,targets,weight=weights if self.temporal els

                mcc = self.mcc(preds,targets).to(self.device)

                acc = self.accuracy(preds,targets)
                precision = self.precision(preds,targets)
                recall = self.recall(preds,targets)

                return loss,[],acc,preds
```

Listing 12: PyTorch Lightning Forward pass method

5.3.3 Temporal-proximity in truth labels

The delta date is calculated by taking a truth date t_t and subtracting along all dates in the label dates $t_l = t_1, t_2, ..., t_n$. The output lists delta times expressing the days since the species was reported. The implementation of the method is shown in Code listing 13

```
1
2  def to_utc(dt):
3      return dt.astimezone(datetime.timezone.utc)
4
5  def from_string(string : str,fmt="%d.%m.%Y"):
6      return datetime.datetime.strptime(string,fmt)
7
8  def create_delta_time(truth :str, obj):
9      truth = from_string(truth,fmt="%Y-%m-%dT%H")
10     return [(to_utc(from_string(x)) - to_utc(truth)).days for x in obj]
```

Listing 13: date time conversion

In each step of the model's loop, the weights are created. Temporal proximity is implemented for both Mean Squared Error (MSE) and Binary Cross-Entropy (BCE) by calculating the sum of weights from each species' history. The implementation of this method can be seen in the code snippet provided in Listing 14. These calculated weights are used in conjunction with the mean squared error or binary cross-entropy to incorporate temporal information into the training.

In the training process, each batch contains a target vector and corresponding dates for the vector. If the output vector size is N = 3, there will be a corresponding list of dates for each of the three species.

The method in Listing 14 calculates the sum of weights based on temporal proximity for each target species. This method incorporates temporal information by assigning weights to the elements of the target vector.

Figure 4.2, mentioned in the Design, visualizes how the weights look using a bell curve representation. This visualization provides an understanding of the distribution of weights based on the temporal proximity of the target values.

```python
def gaussian_function(x, mu,sig):
    return np.exp(-np.power(x - mu, 2.) / (2 * np.power(sig, 2.)))

def calculate_loss_resampling_weight(dates : list = [],mu=0,sig=2000):

    weights = np.ones((len(dates),len(dates[0])))

    for i,target in enumerate(dates):
        sum_w = [gaussian_function(np.array(target[i]),mu,sig) \
                    for i in range(len(target))]

        for j, l in enumerate(sum_w):
            if len(l) == 0:
                sum_w[j] = 1 # Dates should always exist.
            else:
                sum_w[j] = sum(l)/len(l)
        weights[i] = np.array(sum_w)

    return weights
```

Listing 14: temporal proximity gaussian fit

6 Evaluation

This thesis presents a novel method of labelling echosounder data. The echo data labels are composed of every catch message within a perimeter. To investigate whether the catch messages make a good proxy for the abundance in an echo image, we compare it with an already-annotated data set.

Furthermore, the effectiveness of the labelling is investigated. The threshold on the haversine distance can be increased or decreased. A larger radius can potentially give more information about fish abundance in an area. On the other hand, a smaller radius will give a more pinpointed evaluation.

6.1 Sandeel survey

As presented in related works in Section 3.2.1, CRIMAC proposed multiple classification models for the sandeel survey conducted by HI. As HI have an open data policy, we requested the dataset from Kongsberg Maritime. The dataset [1, 2], contains *.bot* files containing seabed estimates from the echosounder. This allowed more detailed information in the seabed indexes. All data in each ping below the seabed index is masked to a specific intensity.

The data is highly-calibrated multi-modal echosounder data. For this thesis, we delimit ourselves to one of the frequencies 128 kHZ, as gridding and aligning of multiple back-scatters gives a higher logic complexity.

Threshold	Number of distinct species
1	73
5	98
10	110
20	130

Table 6.1: Number of distinct species in each threshold label set

6.2 Benchmark setup

The hardware specifications listed in Table **??**, is the compute node used throughout the thesis.

	Hardware information
CPU	Genuine Intel(R) CPU @ 2.00GHz x 72 ; 2 threads per core
RAM	8 * 32 GB (256 GB) Samsung 41E8436D
GPU	1 x GeForce RTX 2070 , 1 x GeForce RTX 2060
OS	Ubuntu 20.04 (Focal Fossa)

6.3 Experimental environment

This section describes the test environment for the experiments done in this thesis. We evaluate the annotation method both through analysis of the resulting labels, as well as through model training.

Each echo example is stabilized through a median filter and seabed segmenting. Seabed segmenting is done using the mentioned .bot file and segmenting all intensities below the seabed to a low intensity of -90 dB.

Furthermore, the examples are cropped to a satisfactory size with regard to the input dimensions of the model. Each patch is of size $512x526$, meaning a patch represents 512 pings, and each ping is 526.

The output heads in the regression models and multi-output heads depend on the threshold's unique species set. Table 6.1 shows the different output heads used on the different thresholds.

6.3.1 Model parameters

We ran all model experiments with a learning rate of 0.001 for 40 epochs. For temporal experiments, we used $\mu = 0$ and $\sigma = 500$ for the Gaussian weights. The threshold for the collation criterion was 1,5,10,20.

Three types of models, *Binary*, *multi*, and *regression* was evaluated.

- Binary Model: The model outputted a single value representing the presence or absence of lesser sand eels.
- Multi Model: The model outputted all species in the label set for the various thresholds tested.
- Regression Model: The model performed regression analysis, predicting the round weight distribution of lesser sand eels and potentially other species.

For every run, the dataset was shuffled and divided into a data split, where the training dataset was 0.8, and the validation and test set was 0.2.

6.4 Annotation experiments

In the initial exploration of the annotation data, an evaluation was done with regard to the catch data. This evaluation aims to understand the labels and obtain information about the label's features. The data for these experiments are both from the labels and from statistical information found during the collation of the labels. For each echo example and each threshold, statistical information is stored, such as the number of DCA messages (label size) collated and the time spent. Moreover, each label contains a round weight of species and the corresponding dates.

6.4.1 Label size

One aspect examined was the number of catch messages found within the thresholds. The experiments had thresholds set at $T = 1, 5, 10, 20$. The impact of these thresholds on the collation criterion was investigated.

Figure 6.1 is a graph depicting the number of catch messages for each threshold. This visualization provides insights into how the choice of threshold affects the magnitude of each label's size per echo example.

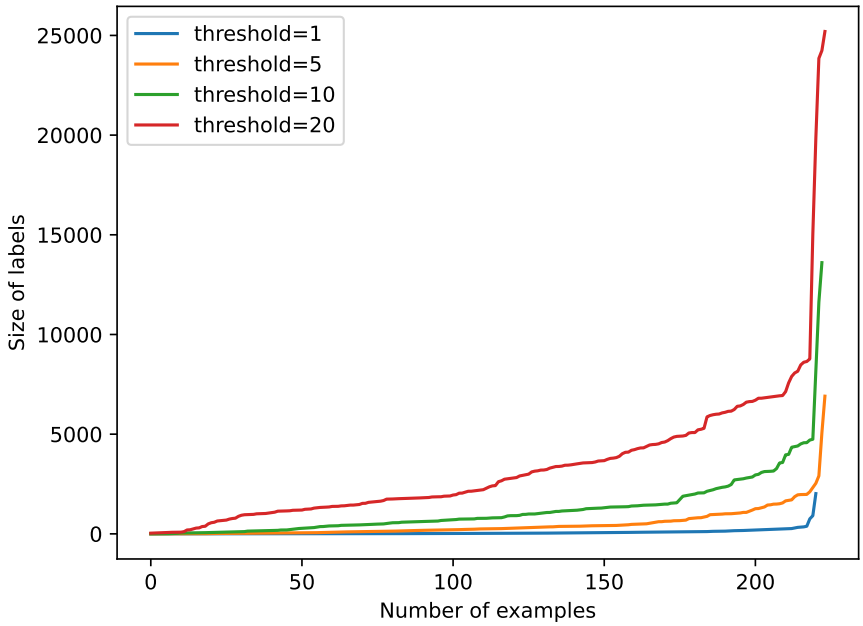

Figure 6.1: Magnitude of catch messages for each threshold T, compared to number of echo examples

6.4.2 Unique species in labels

This experiment concerns itself with the number of unique species found in each label set for the different thresholds mentioned in section 6.4.1. The goal is to understand how the choice of threshold impacts the diversity of species captured by the labels.

The number of unique species present in each label set was determined by evaluating the annotation data using the specified thresholds ($T = 1, 5, 10, 20$). This analysis provides insights into the variety and distribution of species captured at different threshold levels.

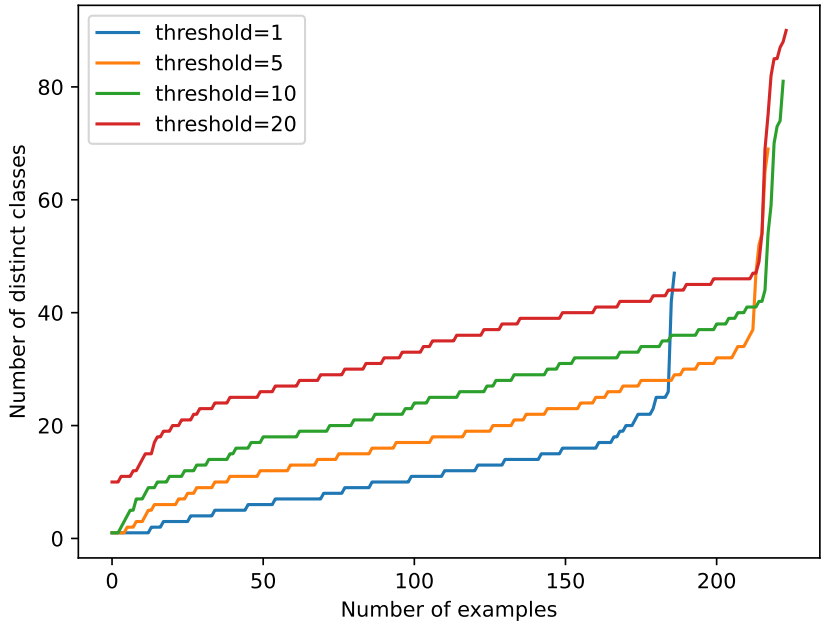

Figure 6.2: Number of different species for each threshold T

6.4.3 Distribution of temporal proximity in data

We examined the distribution of temporal proximity in the data to explore the potential of the weighted loss function. This analysis involved mapping all collated labels to their corresponding potential truth candidates based on date (The date of the echo example).

The resulting distributions were visualized using a histogram with respect to

threshold T. Each spike in the histogram represents a periodical increase in the catch. The spikes are roughly a year apart in days, indicating a recurring pattern in the catch data.

Figure 6.3 displays the histogram depicting the number of species for each threshold value T. This visualization provides insights into the distribution of species and their temporal proximity based on the evaluation of truth dates with label dates.

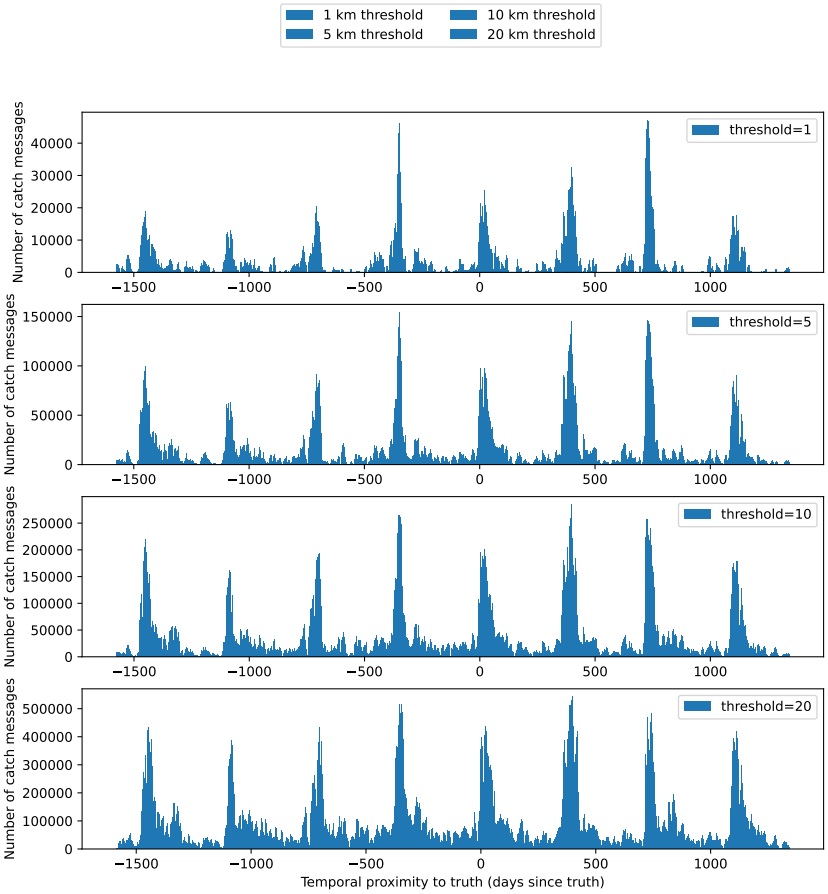

Figure 6.3: Number of different species for each threshold T

The logic behind calculating these distributions is explained in the code listing 15.

```
1
2  def retreive_temporal_distribution(image_dir:str , dir:str='ds/labels_crimac_2021'):
3      threshold_dict = {1:[],5:[],10:[],20:[] }
4      date_dict = {1:[],5:[],10:[],20:[]}
5
6      for label ,threshold in load_dir(dir):
7          for species in label:
8              threshold_dict[threshold].append((label[species]['date']))
9
10     for img_file in os.listdir(image_dir):
11         truth = img_file.split('-')[1]
12
13         for k in threshold_dict.keys():
14             for t in threshold_dict[k]:
15                 for date in t:
16                     date_dict[k].append((to_utc(from_string(date)) \
17                     - to_utc(from_string(truth,fmt="D%Y%m%d"))).days)
18                 date_dict[k].append(create_delta_time(date,t))
```

Listing 15: proximity distribution

6.4.4 Computation Time of Catch and Echo Collation

In this experiment, we aimed to assess the computational overhead associated with calculating the distance matrix for catch and echo collation. The objective was to understand the relationship between the time required for calculating the distance matrix with different thresholds and the size of the found catch messages within the specified region.

To visualize this comparison, a scatter plot was generated. The x-axis represents the time spent calculating the distance matrix, while the y-axis represents the size of the found catch messages within the region. Figure 6.4 provides a visual representation of this comparison, allowing a better understanding of the computational trade-off associated with different threshold values.

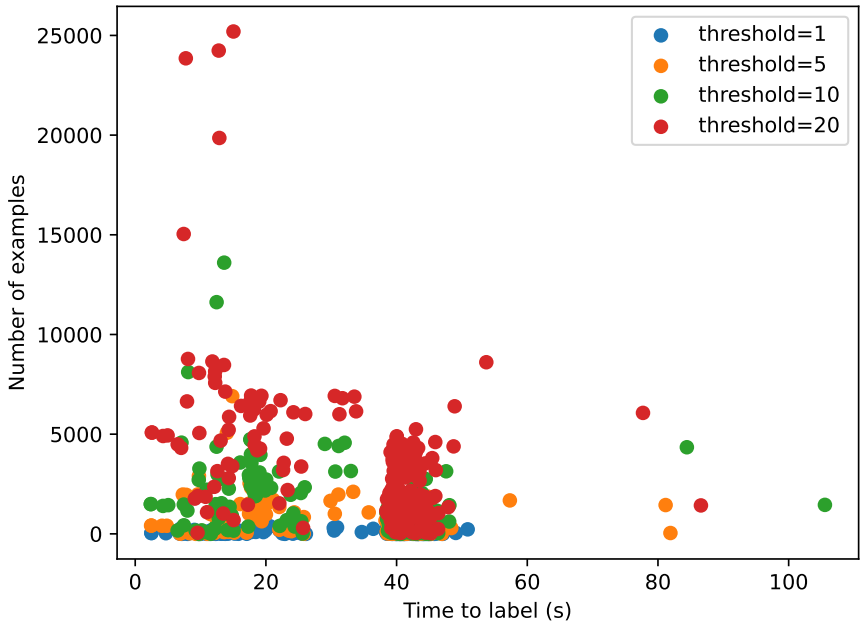

Figure 6.4: Plot expressing computation time compared to the size of found DCA messages.

6.5 Model experiments

This section outlines the experimental methods for our proposed usage of the annotated data in regression and classification tasks. We conducted three primary experiments, and each experiment is aligned with metrics for running the models with and without temporal proximity weighting.

6.5.1 Regression method

In the regression experiment, the objective was to predict an output vector representing the species distribution. To create the target vector, the round weight of each species was summed to obtain the total weight, which was then divided by the sum to normalize the distribution.

The evaluation of the regression models involved the use of two metrics:

Mean Squared Error (MSE) Loss: This metric calculates the average squared difference between the predicted and actual target values in the validation data. It provides an overall measure of the model's performance in terms of the accuracy of its predictions.

Mean Absolute Error (MAE): This metric compute the average absolute difference between the predicted and actual target values in the validation data. It provides a measure of the model's average prediction error, irrespective of the direction of the error (Magnitude).

Figure 6.5 and 6.6 depicts the Loss (MSE) and Mean absolute error of the model's performance on the validation set per epoch. The data is averaged over three runs, with a shuffled dataset and dropout of 0.1 in the model's layer.

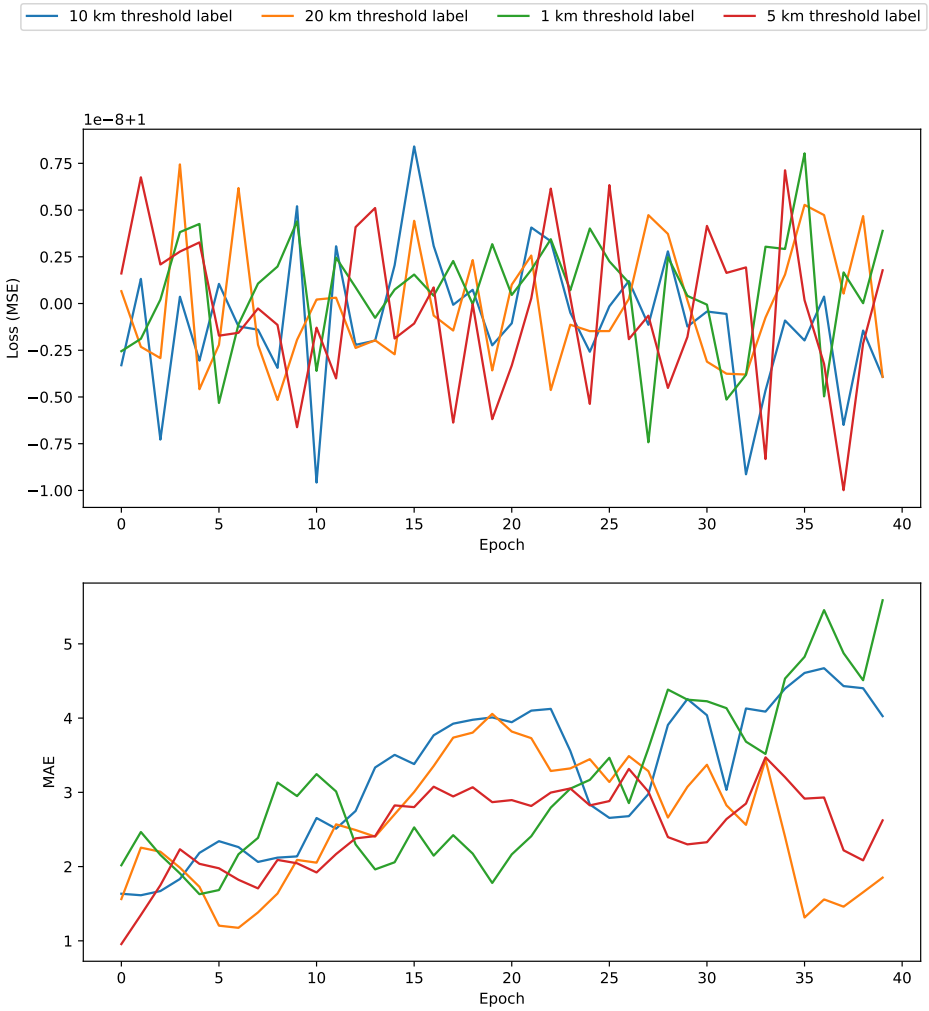

Figure 6.5: Loss and Mean Absolute Error for regression model

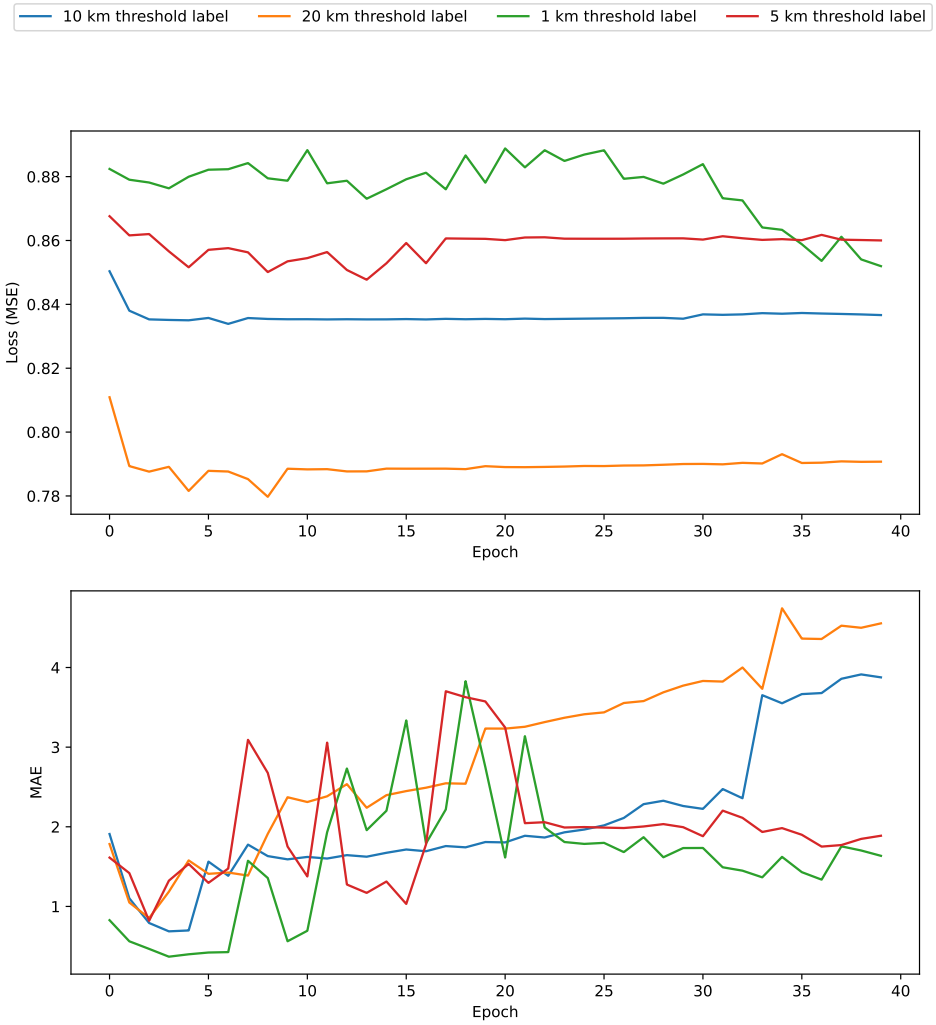

Figure 6.6: Loss and Mean Absolute Error for regression model with temporal proximity

6.5.2 Classification method

Two classification methods were employed and experimented with in this study. The first experiment used a **binary** output vector, where the model labels represented a single species. Specifically, the focus was on classifying *lesser sand eels*.

Figure 6.7 and Figure 6.8 illustrate the model's performance on this classification task.

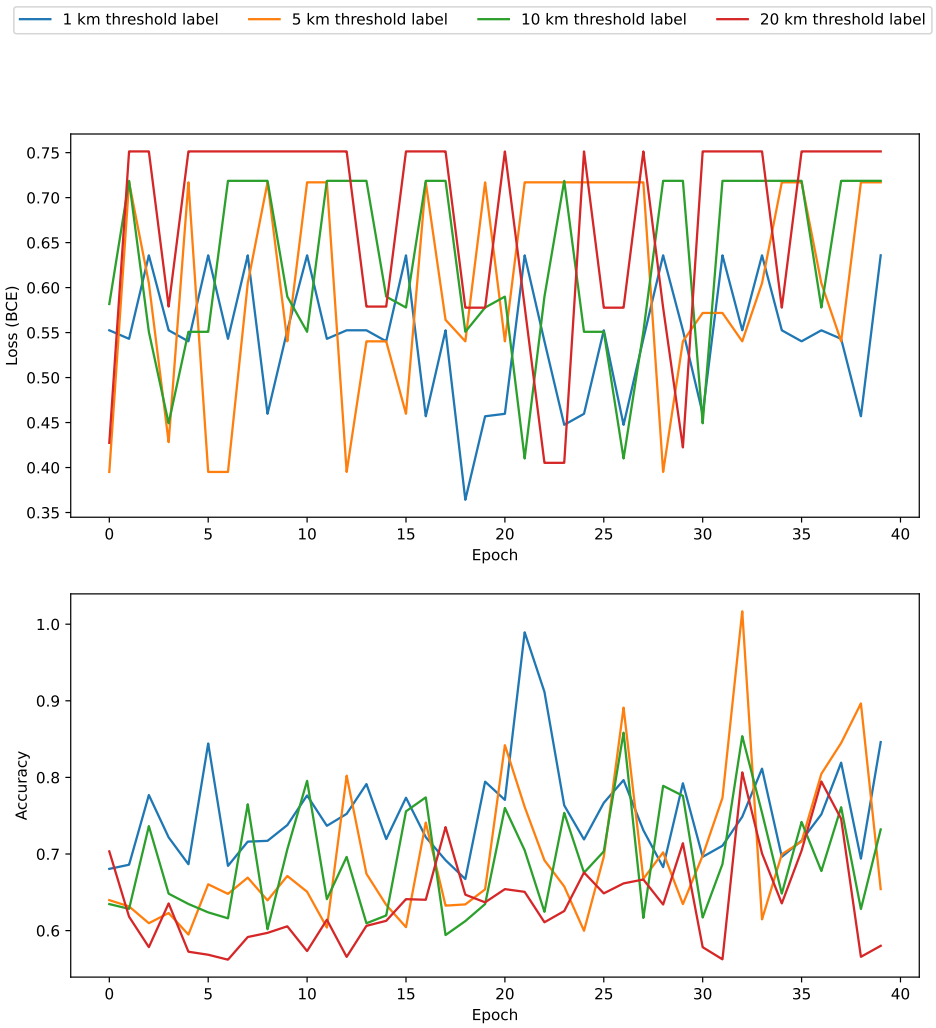

Figure 6.7: Plot without temporal weighting, showing Loss and Accuracy of binary model

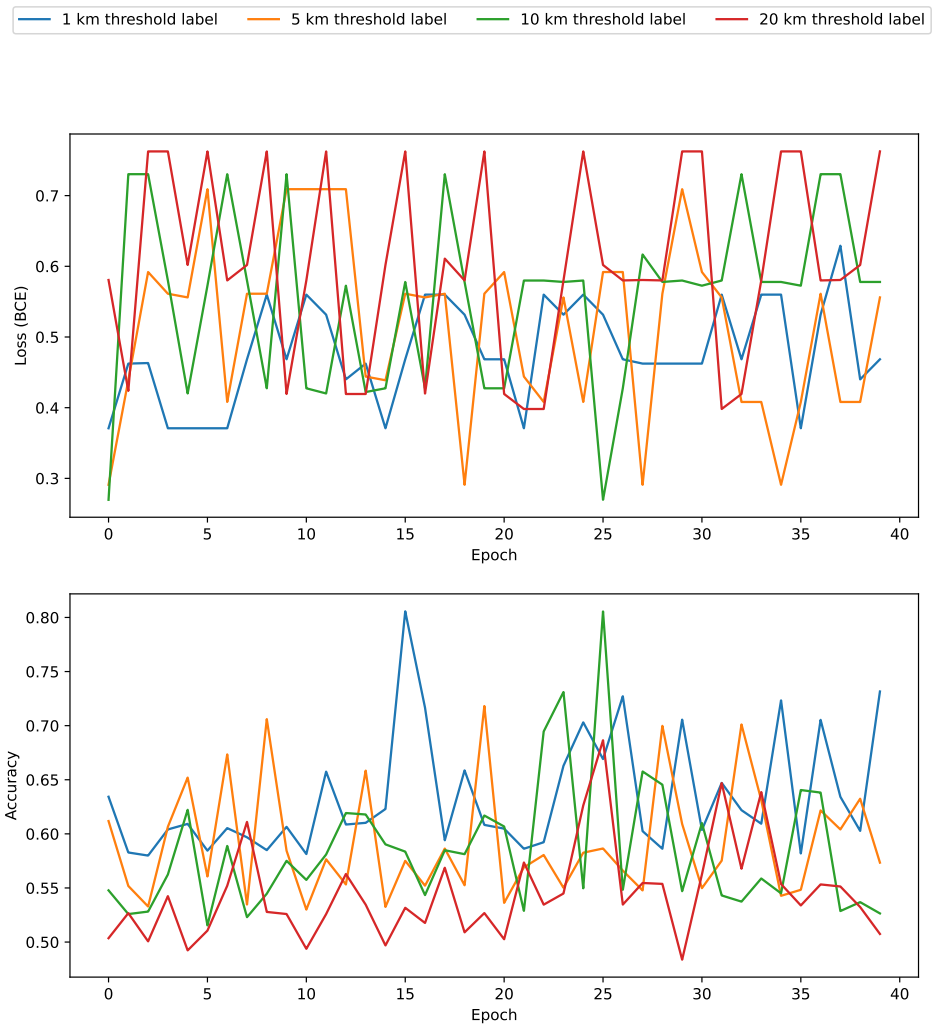

Figure 6.8: Plot with temporal weighting, showing Loss and Accuracy of binary model

A **multi**-classification approach was employed in the second experiment, which utilized all annotations corresponding to the specific threshold value. Table 6.1 outlines the details of the output heads used in this experiment, along with the dimensions of the target vectors for each threshold.

Figure 6.9 and 6.10 illustrate the model's performance on this classification task. The Loss, Binary cross-entropy, and Accuracy were evaluated per epoch on the validation set.

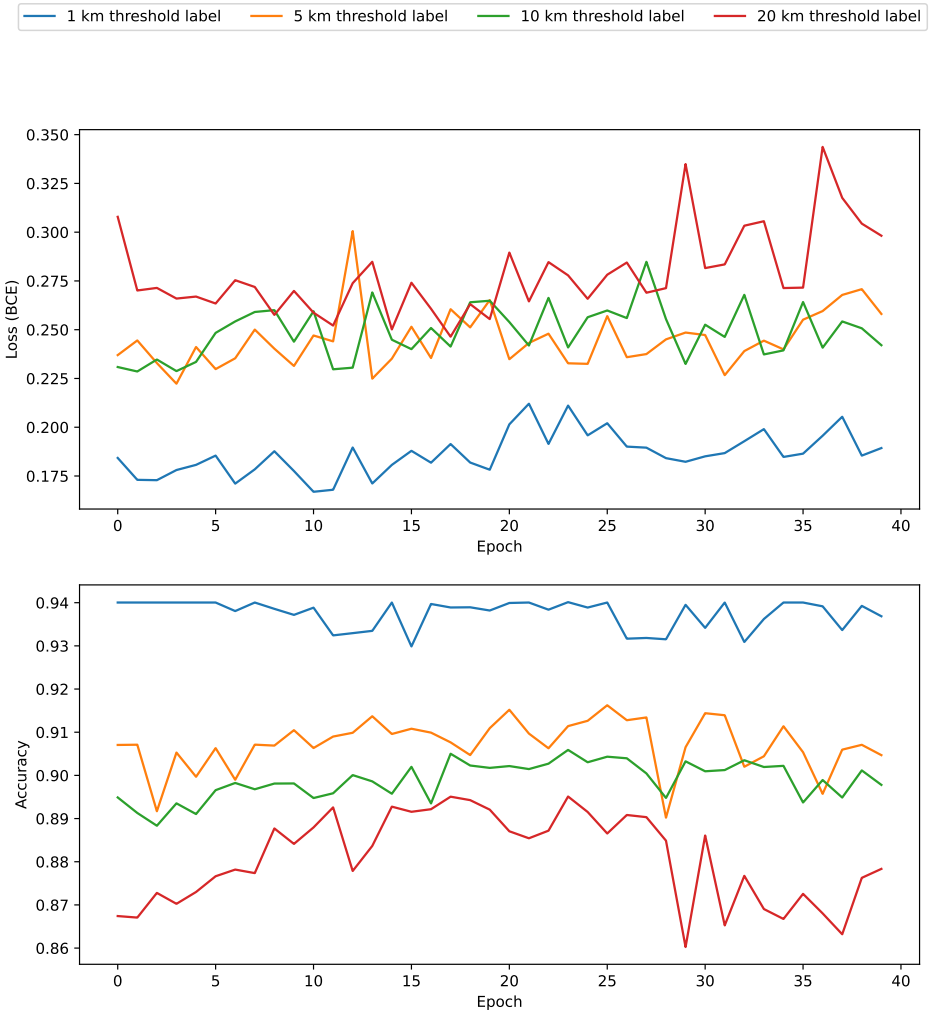

Figure 6.9: Plot without temporal weighting, showing Loss and Accuracy of multi model

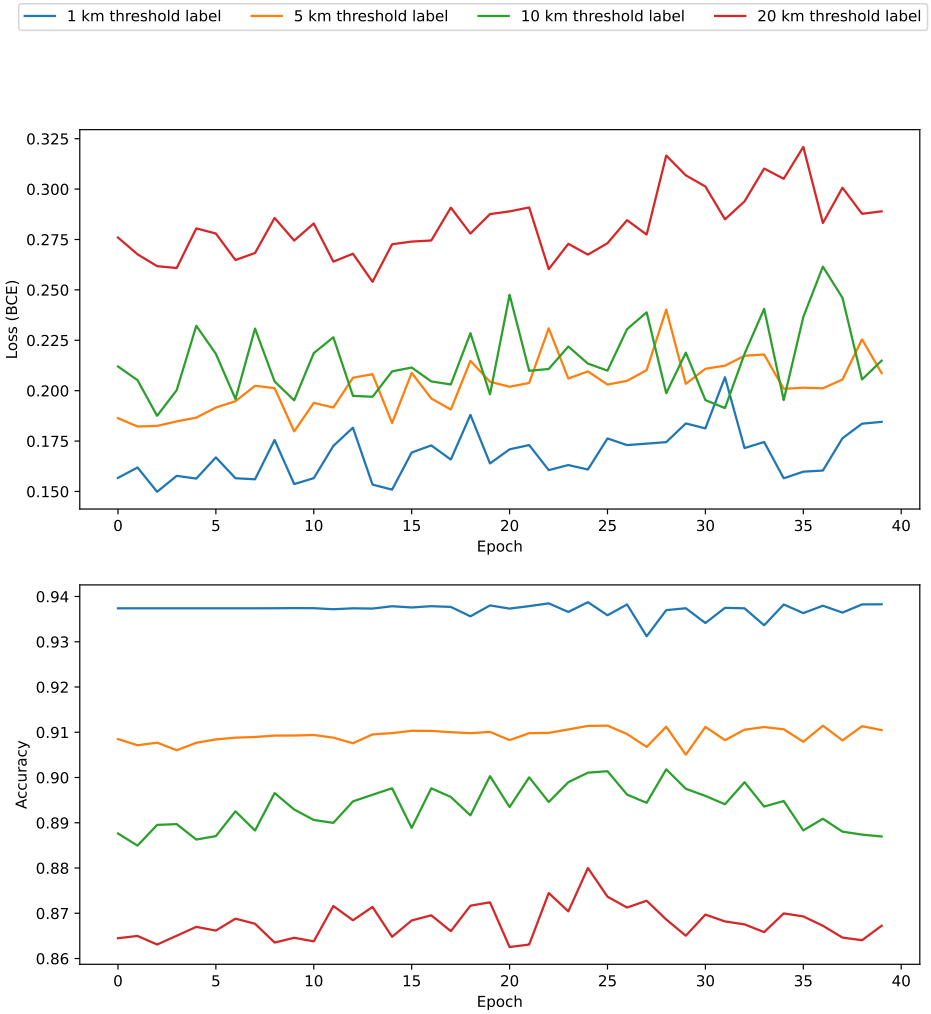

Figure 6.10: Plot with temporal weighting, showing Loss and Accuracy of multi model

7 Discussion

7.1 Preprocessing

Stabilization and removal of unwanted features were one of the requirements of the design. This was done by utilizing the *.bot* files from the sand eel survey dataset. The testing environment did not utilise the functionality of cropping the image based on the bottom indexes but segmented all features below the seabed to an intensity value of −90 dB. By cropping the seabed as presented in figure 5.1, the datasets patches would hold more relevant information yet decrease the size of the dataset.

The sand eel survey data is already calibrated, so the backscatter exhibits a high signal-to-noise ratio. Consequently, noise is not a major concern in this particular case. Therefore, the need for noise filtering during training is minimal. However, noise filtering can become more important in-situ operations where echosounders are not optimally calibrated.

The preprocessing of the .raw files utilized xarray Datasets as the data structure, offering convenient indexing capabilities for dimensions and variables, including auxiliary information.

Moreover, adopting Dask, a xarrays parallelization framework, allows for out-of-core computations, yielding faster reads and writes for the files than traditional file formats. [27, 17].

7.2 Exploring the annotation

This section will first give an analysis of the experiments done. Then move on to discuss the implementation itself.

7.2.1 Label analysis

The experiments done regarding the annotation method are described in section 6.4. The graphs give information about the annotation method.

Firstly, the plot depicted in Figure 6.1 illustrates each annotation's varying sizes (measured by the number of catch messages). The visualization displays a scale factor that corresponds to the set threshold. For instance, a threshold of $T = 20$ is twice as large as $T = 10$. This scaling relationship can be observed among the other thresholds as well.

Secondly, evaluating the number of species found with regard to the threshold radius is another important experiment to conduct. Figure 6.2 shows the distinct classes (species) found. A remark to present is the thresholds $T = 1$ lack of finding catch messages for some labels.

Figure 6.3 visualises the temporal proximity for each example in the echo data. The season of fishing activity can be seen from the yearly period spikes. This suggests that the found labels are likely to be clustered in time, indicating a potential bias in the data.

The temporal clustering of labels within specific periods, such as fishing seasons, can introduce bias in the dataset. This bias may affect the generalizability and representativeness of the model's performance.

Lastly, we investigate the compute time of the parallelized *numba* distance matrix calculation. A scatter plot shows the relationship between compute time and the number of labels found. Most of the processing times and sizes lie in the same quadrant of the plot, but outliers exist. With this, we concluded that it was not the amount of catch messages found that was the compute expensive part of the collation.

7.2.2 Label processing

The processing of labels was implemented by collating echosounder data with positional attributes of DCA messages. The collation was done by creating a distance matrix which was threshold and indexed back to the *xarray* dataset

created in the preprocessing.

As we utilized *numba* as our parallelization method, the parallelization methods offered by xarray became redundant. Although xarray lazy evaluation and parallelizability could potentially provide benefits, the implementation overhead compared to *numba* made a compelling argument. The JIT compiler's embarrassingly parallel functionality is something not to take for granted. An experiment evaluating the two frameworks' speed compared to each other would be interesting.

As the distance matrix calculation filled up the benchmarks cores, parallelisation made some echo examples unsolvable. An assumption made for the collator is that if the shape of the echo ping is greater than 6000, the example will be skipped in the calculation. This was because the benchmark setup, with 72 cores and 256 GB memory, started swapping memory access, leading to a full halt in the process. We also concluded that a watchdog timer was necessary to avoid hanging on the process. Improvements to this will be discussed in future work.

7.3 Using the annotation

7.3.1 Model dataloader

PyTorch data loaders were utilised while implementing the echo and annotation examples. As mentioned in Section 5.3.1, the data loading process was implemented using a PyTorch data loader. This framework offers the advantage of parallelizing batch retrieval from disk, allowing multiple workers to be assigned to load batch files simultaneously.

However, there was an issue with the data loader's compatibility with another parallelizable method. Specifically, the *.zarr* files generated by the preprocessing components were loaded lazily from the disk, meaning that they were only fetched when the data was evaluated (when printed). Unfortunately, the data loader did not cooperate seamlessly with this parallelization approach. As a result, the echo examples were converted to numpy arrays and stored as associated files to ensure proper functioning.

7.3.2 Model analysis

During the fine-tuning task, validation data was fed into the model after each epoch, and metrics were calculated. We had two primary model types: regres-

sion and classification.

Regression model

The regression task used percentages, as presented in 5.2, as targets. Figure 6.5 shows the metrics without temporal proximity. The aforementioned has an MSE circulating around 0 and an increase in MAE after each epoch. The model's Loss function is spiked, and no patterns between the thresholds can be seen. The MAE can be seen as more robust than the MSE, as the model shows far fewer spikes in difference in magnitude.

As the target class lies between 0 and 1, the model regresses fast to this interval, and after the first epoch, all model lies under 1. Whether this is, a method of modeling the catch data remains unanswered. But by the metrics, it can be seen as a possible solution.

The **Regression** model with temporal proximity, illustrated in figure 6.6, shows the thresholds loss values, where $T = 1$ has the highest loss, while the $T = 20$ has the lowest. This is because the amount of *DCA* messages found for $T = 20$ is much higher than $T = 1$ as seen from the temporal experiment in Fig. 6.3. Consequently, the weights have thus a greater scaling factor for lower thresholds.

Classification model

Both classification tasks have a loss (Binary cross-entropy) and Accuracy as primary metrics. The first task, **Binary** illustrated in Figure 6.7, shows the loss and accuracy for each threshold. The lowest threshold, $T = 1$ kilometers also has the lowest loss response and the highest accuracy. This shows that it may be beneficial with a *fine-granularity* in the annotated labels. This is also shown by $T = 20$, where the coarse-granularity labels have the highest loss and lowest accuracy.

The **Multi** classifiers Figure 6.9 illustrates the same behavior as for the binary classification task. However, the model is much more robust to the outliers. The accuracy in the multi-task is higher; $T = 1$ is 94% accurate, and the others lie over 85%. The reason for the high accuracy in the task can be reasoned by the choice of loss function. The BCE is used as objective function, and the model outputs whether the selected species exists in the echo data.

Finally, Figures 6.8 and 6.10 depict the performance of the models where the

loss is scaled by temporal proximity. These weights aim to adjust the difference between predictions and targets based on the dates of the targets. The loss values in the temporal models can look as though they have been normalized or averaged over, but in reality, it is the cause of the down-scaling by the temporal weights.

The **Multi**-head classification model, illustrated in figure6.10, exhibits a distinction between the weighted and non-weighted versions. The temporal model's loss demonstrates a low validation loss, with $T = 1$ being the lowest and $T = 20$ the highest.

In most cases, the model achieves an accuracy of over 50% in all classification tasks. However, this may not accurately assess the model's performance. For instance, consider a scenario where the target vector consists of 100 classes, and the objective is to predict their presence. The model classifies all classes as not present, whereas in reality, 25 of the classes are actually present. This results in 75 true negatives and 25 false positives, leading to an accuracy of 75%.

Therefore, alternative metrics can offer a better evaluation of the model.

Lastly, it is important to consider the temporal weighting implemented in the model. Weights in loss functions are typically used as a countermeasure to class imbalances. Temporal weighting is designed to assign different weights to the differences between predictions and ground truth based on the temporal proximity of the species. However, it is crucial to note that while the assumption behind temporal weighting holds true, the loss function used in the model is typically a mean or sum of all output neurons.

In conclusion, while the temporal weighting implemented in the model may not work as expected for the loss function, it can still provide valuable temporal information to guide the model's training. Further research and analysis are necessary to explore the potential impact and effectiveness of incorporating temporal weighting in a model.

8

Conclusion

In this thesis, we have presented the design and implementation of a deep learning pipeline that utilizes echo data and annotations from collated catch messages. The design of the pipeline is based on theoretical considerations and prior research in the field. The preprocessing and retrieval of echo data involved reading Kongsbergs .raw files and organizing them into labeled multidimensional arrays. The most computationally intensive step involved collating echo examples and catch messages, resulting in large matrices with approximately 9 billion elements per echo example. By employing parallelization techniques with *numba*, this task was completed in an average time of 34 seconds.

For the model architecture, we employed EchoBERT as a base model and made modifications to suit our specific needs. We conducted experiments with multiple different output heads to understand the utilization of our annotations.

The annotation experiments revealed important insights into the characteristics of the annotation method. The analysis of label size, the number of distinct species found, and the temporal proximity of labels provided valuable information about the data. We observed a scaling relationship between the label sizes and the threshold values and noted that certain thresholds exhibited limitations in finding catch messages.

The visualization of temporal proximity based on the echo data highlighted the presence of yearly period spikes, indicating the seasonality of fishing activity.

This temporal clustering of labels raised concerns that labels likely clustered in time, introducing a potential bias in the data. Based on the experiments, it became evident that accuracy alone might not be a sufficient metric for evaluating the classification model's performance. The presence of imbalanced classes can lead to misleading accuracy values.

This paper implemented our novel method, temporal proximity weighting of loss functions. The idea was to leverage the timestamps in the annotation to train the model based on proximity in time and proximity in the target objective. By doing so, the idea was to lower the loss in the distant neurons in time, while the neurons close in time gained a higher loss. Even though the temporal weighting did not seem fit in this case, it is a sound idea giving real-world data a dimension of uncertainty in the measurements. Further research incorporating temporal weights in a model is needed, and we suggest some promising directions in Section 8.1.

8.1 Future work

This section describes improvements that can be made to improve the pipeline further. We divide the future work section as per the requirements from Section 4.1.

8.1.1 Processing layer

First of all, the volumetric backscatter coefficient found in the echo data, s_v, may be stacked to multi-modal frequency echograms. In the thesis, we used one of the frequencies captured, 128 kHz. CRIMAC's work on preprocessing indicates that they have stacked multiple frequencies into their CNNs in order to capture features over multiple frequencies. They had their main frequencies as 200kHz in their works [1, 2] because of the target strength of sandeel and for maximization of signal-to-noise ratio. Using different or more frequencies, the model may capture more of the sand eels features.

Furthermore, the collation criterion required us to delimit the collation to samples below 6000 pings, with a further investigation regarding resource starvation and memory consumption. For example, splitting the echoes into smaller vectors may benefit the computation as not as much data must be held in memory.

Another experiment could also be done where xarrays out of core computation are compared to parallelizing with Numba. The out-of-core computation may

contribute to minimizing memory starvation.

Finally, more precise patching needs to be done. As we did not crop the seabed away, many of the patches may contain the mask value intensity of −90. First, cropping the seabed and, afterward, segmenting the patches will contribute to capturing more informational data.

8.1.2 Optimizing the model

Even though the model may not have performed as expected, many propositions have been constructed. With *temporal proximity* weighting being, to our knowledge novel, it may have immense potential. This technique offers a unique way of expressing uncertainty in labels. For instance, one possible application is to incorporate it as a standalone loss function, where the target and temporal loss are summed rather than multiplied. Alternatively, it can be employed as a regularizer term or optimizer. The objective of the temporal proximity mechanism was to encourage the model to assign greater importance to more recent samples during backpropagation compared to outdated labels. An optimizer would then strive to align the gradients with the desired objective, in this case, emphasizing learning in proximity.

Additionally, we propose two additional experiments to evaluate the model. Firstly, we can better understand the model's performance by utilizing the *F1 score* on the test set, which considers both precision and recall.

Comparing the annotated labels predicted labels, or probabilities with the sandeel survey annotations could provide valuable insights into the model's performance in real-world scenarios.

Bibliography

[1] C. Choi, M. Kampffmeyer, N. O. Handegard, A.-B. Salberg, and R. Jenssen, "Deep Semisupervised Semantic Segmentation in Multifrequency Echosounder Data," *IEEE Journal of Oceanic Engineering*, pp. 1–17, 2023. Conference Name: IEEE Journal of Oceanic Engineering.

[2] C. Choi, M. Kampffmeyer, N. O. Handegard, A.-B. Salberg, O. Brautaset, L. Eikvil, and R. Jenssen, "Semi-supervised target classification in multifrequency echosounder data," *ICES Journal of Marine Science*, Aug. 2021. Number: fsab140.

[3] P. Denning, D. Comer, D. Gries, M. Mulder, A. Tucker, A. Turner, and P. Young, "Computing as a discipline," *Computer*, vol. 22, pp. 63–70, Feb. 1989. Conference Name: Computer.

[4] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design Science in Information Systems Research," *MIS Quarterly*, vol. 28, no. 1, pp. 75–105, 2004. Publisher: Management Information Systems Research Center, University of Minnesota.

[5] "Utvikle systemer for beslutningstøtte i fiskeflåten (DataFangst)."

[6] R. C. Gonzalez, R. E. Woods, and B. R. Masters, "Digital Image Processing, Third Edition," *Journal of Biomedical Optics*, vol. 14, no. 2, p. 029901, 2009.

[7] R. C. Gonzalez and R. E. Woods, *Digital image processing*. New York, NY: Pearson, 2018.

[8] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[9] H. Måløy, "EchoBERT: A Transformer-Based Approach for Behavior Detection in Echograms," *IEEE Access*, vol. 8, pp. 218372–218385, 2020. Conference Name: IEEE Access.

[10] "Understanding LSTM Networks – colah's blog."

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," Dec. 2017. arXiv:1706.03762 [cs].

[12] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," May 2019. arXiv:1810.04805 [cs].

[13] "Elektronisk rapportering fra fiskefartøy."

[14] "Åpne data: elektronisk rapportering (ERS)."

[15] O. Brautaset, A. U. Waldeland, E. Johnsen, K. Malde, L. Eikvil, A.-B. Salberg, and N. O. Handegard, "Acoustic classification in multifrequency echosounder data using deep convolutional neural networks," *ICES Journal of Marine Science*, vol. 77, no. 4, pp. 1391–1400, 2020. Number: 4.

[16] E. Johnsen, R. Pedersen, and E. Ona, "Size-dependent frequency response of sandeel schools," *ICES Journal of Marine Science*, vol. 66, pp. 1100–1105, July 2009.

[17] S. Hoyer and J. Hamman, "xarray: N-D labeled Arrays and Datasets in Python," vol. 5, p. 10, Apr. 2017. Number: 1 Publisher: Ubiquity Press.

[18] "CRIMAC-classifiers-bottom," Dec. 2021. original-date: 2021-02-05T12:22:32Z.

[19] "CRIMAC-classifiers-bottom/BottomDetectionAlgorithms.md at main · CRIMAC-WP4-Machine-learning/CRIMAC-classifiers-bottom · GitHub."

[20] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," May 2015. arXiv:1505.04597 [cs].

[21] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, Feb. 2009.

[22] S. K. Lam, A. Pitrou, and S. Seibert, "Numba: A llvm-based python jit compiler," in *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, pp. 1–6, 2015.

[23] C. C. Wall, R. Towler, C. Anderson, R. Cutter, and J. M. Jech, "PyEcholab: An open-source, python-based toolkit to analyze water-column echosounder data," *The Journal of the Acoustical Society of America*, vol. 144, p. 1778, Sept. 2018.

[24] W. Falcon and T. P. L. team, “PyTorch Lightning,” Apr. 2023.

[25] I. Loshchilov and F. Hutter, “Decoupled Weight Decay Regularization,” Jan. 2019. arXiv:1711.05101 [cs, math] version: 3.

[26] L. N. Smith and N. Topin, “Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates,” May 2018. arXiv:1708.07120 [cs, stat].

[27] X. . D. Out-of Core and l. a. i. P. w. p. on, “xray + dask: out-of-core, labeled arrays in Python,” June 2015.

DCA messages

Table 1: DCA Table

Relevant år	2023
Meldingsår	2023
Meldingstype (kode)	DCA
Meldingstype	Detaljert Fangst og aktivitetsmelding
Meldingsnummer	1
Meldingsversjon	1
Sekvensnummer	;
Melding ID	2776531
Meldingstidspunkt	01.01.2023
Meldingsdato	01.01.2023
Meldingsklokkeslett	00:01
Radiokallesignal (ERS)	LCUF
Fartøynavn (ERS)	Prowess
Registreringsmerke (ERS)	H-2-BN
Fartøynasjonalitet (kode)	NOR
Fartøygruppe (kode)	N
Fartøygruppe	Norsk
Kvotetype (kode)	1
Kvotetype	Normalt fiske

Aktivitet (kode)	STE
Aktivitet	Steaming
Havn (kode)	;
Havn	;
Havn nasjonalitet	;
Starttidspunkt	;
Startdato	;
Startklokkeslett	;
Startposisjon bredde	;
Startposisjon lengde	;
Hovedområde start (kode)	;
Hovedområde start	;
Lokasjon start (kode)	;
Sone (kode)	;
Sone	;
Områdegruppering start (kode)	;
Områdegruppering start	;
Havdybde start	;
Stopptidspunkt	;
Stoppdato	;
Stoppklokkeslett	;
Varighet	;
Fangstår	;
Stopposisjon bredde	;
Stopposisjon lengde	;
Hovedområde stopp (kode)	;

Hovedområde stopp	;
Lokasjon stopp (kode)	;
Områdegruppering stopp (kode)	;
Områdegruppering stopp	;
Havdybde stopp	;
Trekkavstand	;
Pumpet fra fartøy	;
Redskap FAO (kode)	;
Redskap FAO	;
Redskap FDIR (kode)	;
Redskap FDIR	;
Redskap - gruppe (kode)	;
Redskap - gruppe	;
Redskap - hovedgruppe (kode)	;
Redskap - hovedgruppe	;
Redskapsspesifikasjon (kode)	;
Redskapsspesifikasjon	;
Redskap maskevidde	;
Redskap problem (kode)	;
Redskap problem	;
Redskap mengde	;
Hovedart FAO (kode)	;
Hovedart FAO	;
Hovedart - FDIR (kode)	;
Art FAO (kode)	;
Art FAO	;

Art - FDIR (kode)	;
Art - FDIR	;
Art - gruppe (kode)	;
Art - gruppe	;
Art - hovedgruppe (kode)	;
Art - hovedgruppe	;
Sildebestand (kode)	;
Sildebestand	;
Sildebestand - FDIR (kode)	;
Rundvekt	;
Individnummer	;
Kjønn (kode)	;
Kjønn	;
Lengde	;
Omkrets	;
Spekkmål A	;
Spekkmål B	;
Spekkmål C	;
Fosterlengde	;
Granatnummer	;
Fartøy ID	2015067973
Registreringsmerke	H 0002BN
Radiokallesignal	LCUF
Fartøynavn	PROWESS
Fartøykommune (kode)	4601
Fartøykommune	BERGEN

Fartøyfylke (kode)	46
Fartøyfylke	Vestland
Største lengde	60,2
Lengdegruppe (kode)	5
Lengdegruppe	28 m og over
Bruttotonnasje 1969	1612
Bruttotonnasje annen	;
Byggeår	1988
Ombyggingsår	;
Motorkraft	2250
Motorbyggeår	1988
Fartøymateriale (kode)	STÖL
Bredde	11
Fartøy gjelder fra dato	01.01.2020
Fartøy gjelder til dato	31.05.2023
Fartøyidentifikasjon	PROWESS - LCUF - H 0002BN
Fartøylengde	60,2