UNIVERSITY OF BERGEN
DEPARTMENT OF INFORMATICS

Selecting Maximally Informative Frequency Subsets for Acoustic Surveys

Author: Knut Thormod Aarnes Holager

Supervisors: Ketil Malde, Nils Olav Handegard

UNIVERSITETET I BERGEN
Det matematisk-naturvitenskapelige fakultet

June, 2022

Abstract

In this thesis, we sought the most informative subset of frequencies to be utilized when classifying sandeel in acoustic data. The intention was to help identify an optimal choice of frequencies if the choice of transducers were restricted by, for example, price or carrying capacity in autonomous vessels. To measure the information lost, we started by producing pseudo labels with an existing automatic acoustic classifier trained to identify sandeel. Then we trained new classifiers based on the same architecture on the same data, but varying subsets of frequencies were used. We could then measure how well these new models could reconstruct the pseudo labels. The F1-score of the highest performing subset, per subset size, increased from a size of one frequency in use (0.34) to two (0.46), and then drastically to three (0.65), after which only marginal improvements were seen. In particular, the subset containing *18kHz*, *38kHz*, and *200kHz* achieved close to the same performance as using the complete set of six frequencies (0.67). Furthermore, the three frequencies mentioned exhibit unique performance compared to the other subsets of equal size.

Acknowledgements

I would foremost like to express my gratitude towards my friends and fellow master's students. Hans Martin Johansen, Mathias Madslien, Halvor Barndon, Emir Zamwa, Johanna Jøsang, John Isak Villanger, and Gunnar. Thanks to you all for the comradeship, both in the reading room and outside the university. For making each day of this degree enjoyable and supporting me throughout these two years. Particularly to my fellow sandwich makers, whose ignorance of conventional mayonnaise limits, motivated me to continually hit the gym.

Thanks to my supervisor Ketil Malde and co-supervisor Nils Olav Handegard for helping me formulate my thesis and providing invaluable feedback. Gratitude to all the people who provided support from the University of Bergen and IMR, particularly Ibrahim Umar and Tomasz Furmanek, for without your expert knowledge and willingness to help, this thesis would not have been possible. Finally, a big thanks to my family and girlfriend for your unfaltering support.

Knut Thormod Aarnes Holager
01 June, 2022

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter will present the context of the research conducted in this thesis and the motivation and focus of the work. Finally, the research question will be stated, followed by a short description of each chapter.

1.1 Marine Advisory Work

The International Council for the Exploration of the Sea (ICES) is an intergovernmental science organization which is the primary provider of advice on marine ecosystems to the governments and international bodies that manage the North Atlantic Ocean and adjacent seas. It consists of over 6000 marine scientists, over 300 institutes, and 20 member countries, including Norway. Advice is formulated by expert groups who work towards a better understanding of marine ecosystems and their sustainable use [17]. The Norwegian Institute of Marine Research (IMR) is a significant contributor, and one of its activities is to give input to these advices through research and monitoring. An important research area is the monitoring of a species known as the *lesser sandeel*, which is the area to which this thesis seeks to contribute [42].

1.2 Why the Lesser Sandeel?

The lesser sandeel (*Ammodytes marinus*), hereby just *sandeel*, is a small pelagic[1] fish which resides in sandy-bottomed coastal and shallow ocean waters and feeds mainly on plankton. It plays a key role in the North Sea, as it has a critical function in the marine ecosystem as forage fish[2] and as the catch for fisheries. Due to the high level of predation, it lives large parts of its life buried in a sandy seabed, but during the spring feeding season, adults emerge each dawn to create large schools in the upper pelagic layer. Historical data show that changes in their abundance cause bottom-up effects in the ecosystem, causing, for example, breeding failure among several species of seabird [21].

The North Sea is under pressure from various factors, including fishing, coastal construction, maritime transport, oil and gas exploration and production, tourism and recreation, navigation dredging, aggregate[3] extraction, military activities, and wind farm construction [18]. Because of the importance of sandeel, the Norwegian IMR has conducted annual acoustic trawl missions since 2005 in sandeel areas of the North Sea [21]. The goal is to monitor the sandeel stock and create input and data to help the ICES make scientifically-backed advice, one of which is recommendations for fishing quotas [20].

Figure 1.1: *A sandeel buried in the sand.*

Credit: Original image by Mandy Lindeberg [7]

[1]Pelagic fish inhabits the pelagic zone of ocean or lake waters – living neither close to the bottom nor near the shore.

[2]Forage fish are small pelagic fish which are preyed on by larger predators.

[3]Raw materials such as gravel, crushed stone, or sand which are obtained from natural sources.

1.3 IMRs Acoustic Trawl Surveys

IMRs acoustic trawl surveys combine acoustic data from echo sounders, and biological data from trawl catches. When fisheries conduct acoustic surveys of fish, they use *echo sounders* to detect and observe targets in the water remotely. Echo sounders are a special variety of *sonar*, where the acoustic beam produced by a *transducer* is directed vertically downwards from the measuring platform at a set frequency [52]. The echo sounders in use by the IMR usually capture acoustic data at multiple frequencies in parallel to maximize the information acquired [25]. The scrutiny of the data is done through the use of a post-processing software called Large Scale Survey System (LSSS), where *acoustic target classification* is done *manually* with the biological samples as aid [24]. Afterward, the output LSSS-data and biological samples are further processed to estimate the fish abundance and age distribution to support ICES advice [22].

1.4 Acoustic Classification of Sandeel

Acoustic target classification is a significant challenge in most acoustic surveys, and the sandeel is known to be difficult to classify [20, 21]. The earlier mentioned Norwegian sandeel surveys use the *200kHz* frequency to delineate schools, while the *18kHz* and *38kHz* are the most important frequencies for classification when measurements were taken at *18kHz*, *38kHz*, *120kHz*, and *200kHz* [20]. Some earlier works have had trouble identifying sandeel at *38kHz* and *120kHz* [13, 30, 39], while some have had success using combinations of the four frequencies *18kHz, 38kHz, 120kHz, and 200kHz* to specifically separate sandeel from herring and mackerel [38].

Brautaset et al. [3] successfully applied deep learning methods to automatically classify sandeel in multi-frequency acoustic data from the Norwegian sandeel surveys. They extracted and utilized the four frequencies *18kHz*, *38kHz*, *120kHz*, and *200kHz* from the original data, but a problem within deep learning is that feature importance can be difficult to interpret. Hence, it is hard to establish precisely what frequencies contributed the most to their results.

1.5 Unmanned Vehicles in Marine Science

In 2019 Verfuss et al. [55] reviewed the current status of unmanned vehicles suitable for monitoring marine life. The different types of vehicles can operate stationary or moving, on the ocean surface, as aerial or submerged. They can either be remotely controlled, autonomous, or a combination of both. Unmanned vehicles can use a wide array of sensors, but will often have restrictions on what sensors it can carry. For example, the number or types of transducers installed on acoustic platforms can quickly become a structural problem as the size and weight of a transducer drastically increases at lower frequencies[4]. Many of the vessels reviewed are commercially available, which has led to a growth in number and use. The increase in unmanned vehicles and the possibility of gathering more complex data will lead to an exponential rise in the data collected in marine science [31]. This increases the need to make automated programs, or better tools to aid in the current manual data processing, which today is a major chokepoint. However, the nature of the data gathered may change, possibly in decreasing quality, as humans will have fewer or no opportunities to actively engage in the autonomous information gathering processes. This change in information gathering raises the need to optimize the quality of the data gathered by the unmanned vessels.

During the summer of 2020, an unmanned vessel was tested in Årdalsfjorden in Norway which involved a kayak with an electric motor, and one 200kHz echosounder was installed to measure *sprat* abundance. Earlier surveys had indicated that large numbers of sprat live close to the surface, where the traditional research vessels have an acoustic blind zone[5]. The kayak survey showed that the small unmanned vessel managed to measure high densities of sprat in the blind zone. It was also less prone to scare away the fish as the kayak produced significantly less noise, and its size allowed access to shallower waters. The result was positive for the continued use of unmanned vessels, but the manned vessels are still needed for biological samples [23].

[4]Two sample transducers sold by Kongsberg Maritime for fisheries at the frequencies 18kHz and 333kHz weigh respectively 85kg and 2kg [32].

[5]Echosounder are usually mounted on the bottom of the vessel, creating a blind zone up to the surface. Often on a retractable keel to get underneath a layer of bubbles that can be detrimental to the echosounders performance [26].

1.6 Research Question

This thesis aims to create scientific advice concerning which echo sounders should be installed on lightweight unmanned vessels, regarding classification of sandeel in acoustic data. The unmanned vessels may have reduced carrying capacity or other limitations for echo sounders, and the choice of which to install to maximize the information collected is essential. To achieve this, we will expand upon the work of Brautaset et al. [3] and measure the information stored in all possible subsets of the frequencies contained in the original data from the Norwegian sandeel surveys in the year 2018 and 2019: *18kHz*, *38kHz*, *70kHz*, *120kHz*, *200kHz*, and *333kHz*. The research question is stated as follows:

> ”***As lightweight unmanned vessels may not have the capacity to carry all the six echo sounders the IMR usually deploys on the Norwegian sandeel surveys, which ones should be prioritized with the regard to classifying sandeel in multi-frequency acoustic data?***”

1.7 Chapter Overview

This section describes the outline of the thesis and, in short, the content of each chapter.

- **Chapter 2 - Background:** Describes the concepts used in this thesis. First marine acoustics in section 2.1, then machine learning and artificial neural networks in the following sections.
- **Chapter 3 - Basis and related work:** Describes the basis work for which this thesis seeks to further expand upon in section 3.1, and details more recent related research in section 3.2.
- **Chapter 4 - Material and Methods:** Describes the approach to answering the research question. First, our data and the tools utilized, then how the labels were produced combined with data generation, and finally, our experiment.
- **Chapter 5 - Results:** Describes the results from the experiment, which starts with an evaluation of the training process. Then the results are summarized, followed by different findings in the results for our analysis.

- **Chapter 6 - Discussion:** Analyses the results and discusses their implication.
- **Chapter 7 - Conclusion and Future Work:** Describes the answer to the research question, summarizes key findings, and states recommendations for future work.

Chapter 2

Background

This chapter introduces the concepts used during the experiments, within both acoustics and machine learning.

2.1 Acoustics

The echo sounder consists of a *transmitter* that produces a burst of electrical energy at some set frequency. Then a *transducer* receives the output from the transmitter and converts it to an acoustic signal that is propagated through the water, which is also called a *ping*. This forms a directional beam akin to the light from a handheld flashlight. Targets in the water *backscatter/reflect* parts of the energy back towards the transducer. The transducer detects the backscattered sound or *echo*, and the sound is converted back to electric energy as the received signal and is further amplified. The time elapsed when the signal is received determines the range to the target [52].

Pings are usually represented as columns in a 2-dimensional image, also called an *echogram*, with range along the y-axis and time of ping along the x-axis. The columns represent how the acoustic reflections vary for each ping. Any targets detected in the ping will be visualized as a mark in the echogram, usually with different colors depending on the echo strength. In multi-frequency sonars, individual echograms are produced in parallel for each frequency in use, and this is visualized in figure 2.2. Because the echograms are constructed as time × range, the vertical magnitude of a mark indicates the height of the target. At the same time, the horizontal position illustrates changes in *time* if the echosounder is stationary or in *space* if moving. When moving, the echogram thus represents a vertical cross-section of the water column as the transducer is in motion through the water at constant speed in one direction [52].

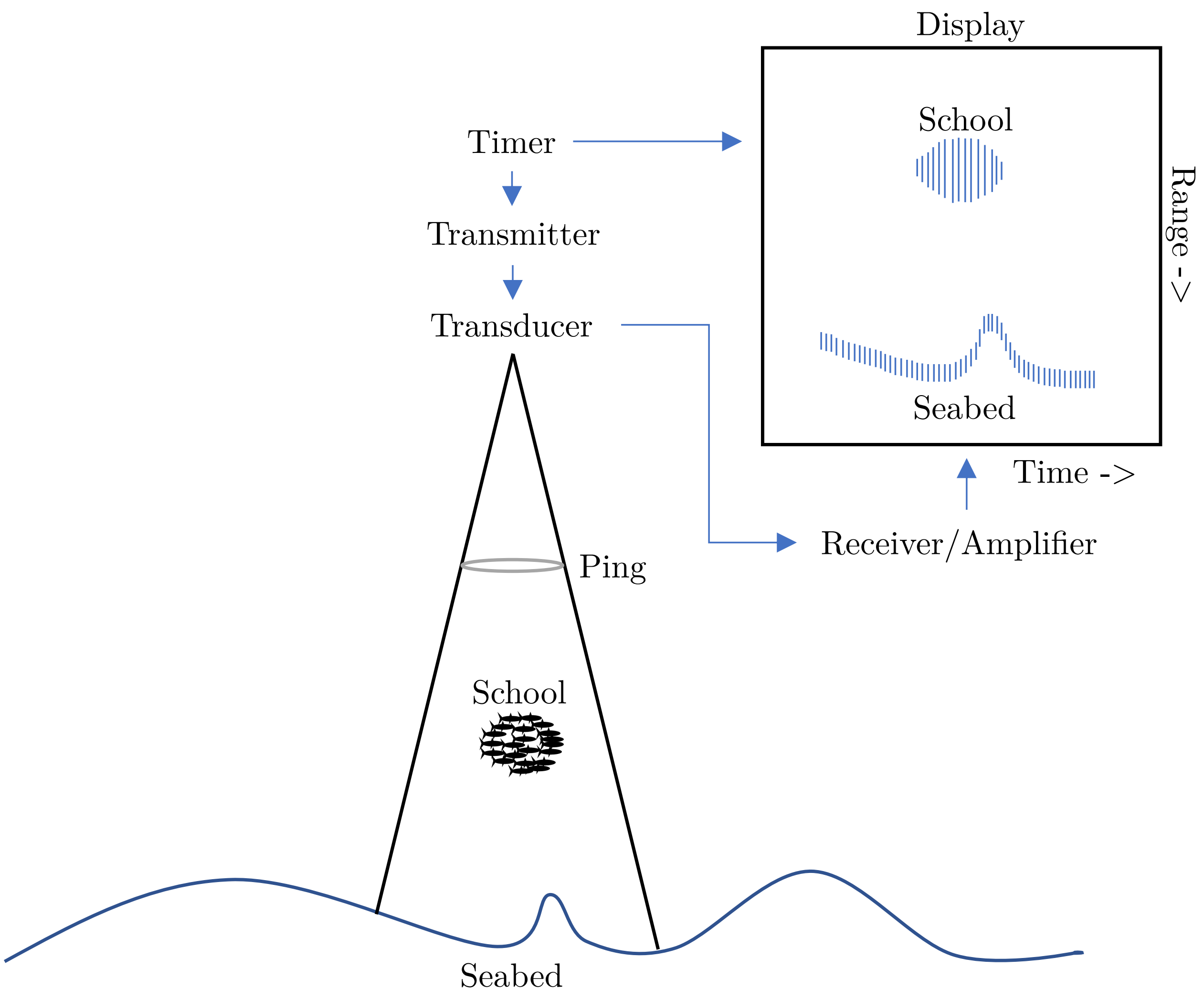


Figure 2.1: *Concept of an echosounder: Pings generate echoes from a school of fish and the seabed, and the echoes are displayed in an echogram.*

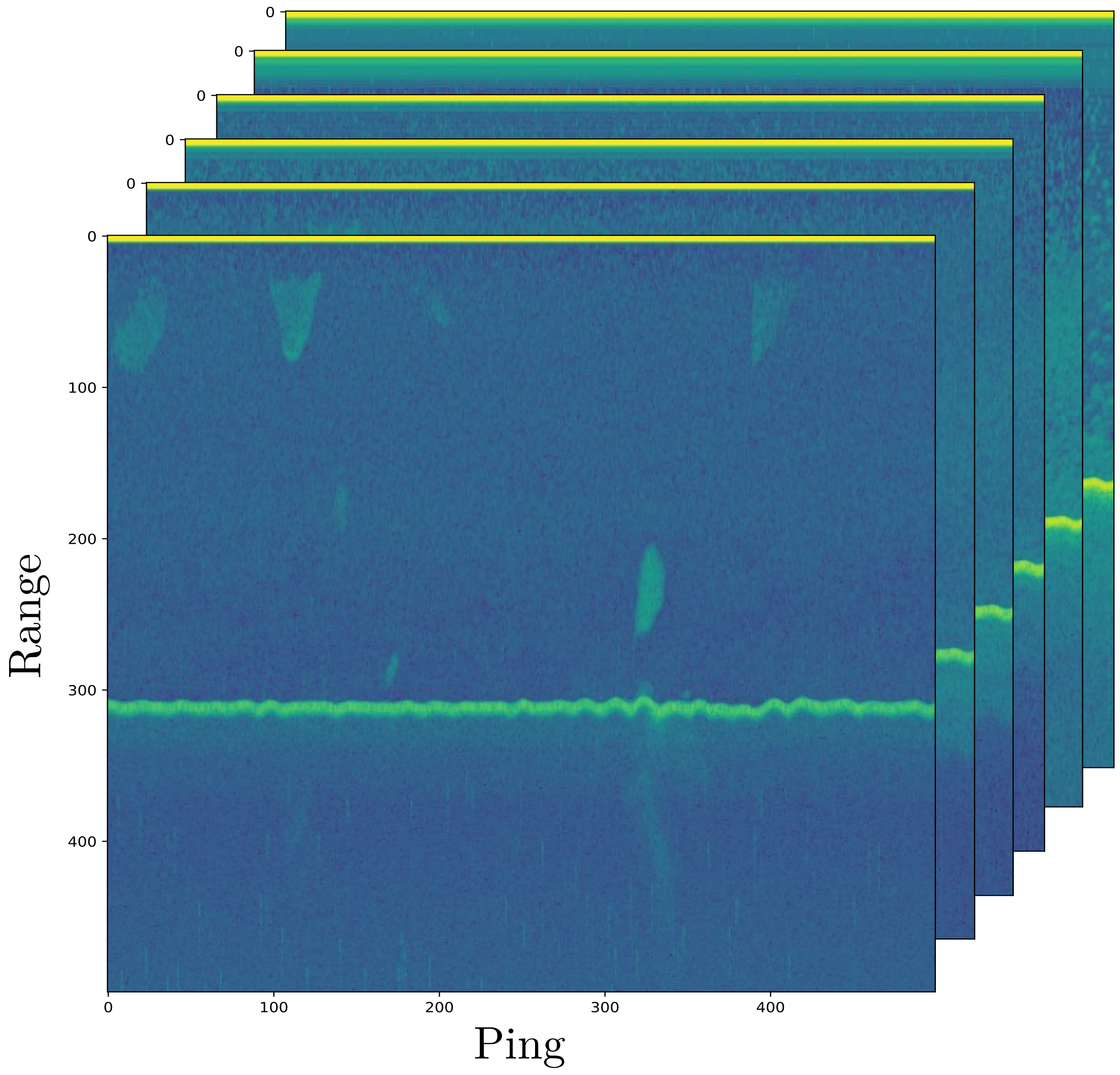


Figure 2.2: *Example echograms from multiple frequencies.*

2.1.1 Acoustics and Fish

To measure the force of backscattered sound received from a target, the backscattered cross-section σ_{bs} or the target strength (TS) is used. They are defined as:

$$\sigma_{bs} = r^2 \frac{I_b}{I_i}, \tag{2.1}$$

$$\text{TS} = 10\log_{10}(\sigma_{bs}), \tag{2.2}$$

where I_b is sound intensity backscattered from the target, I_i is the intensity of the ping at some arbitrary distance (usually 1m), and r is the distance away from the transducer(σ_{bs} in units m^2). σ_{bs} can vary greatly depending on the frequency used, the composition, angle and shape of the target, absorption through sound being converted to heat, and several other factors as described in Simmonds and MacLennan [52].

2.1.2 The Volume Backscattering Coefficient

Individual targets in some sampled volume may be small and plentiful, resulting in their echoes combining and forming a continuous backscattered signal with varying amplitude. Single targets are no longer possible to distinguish, but the signal itself is a measure of the biomass in the water column. This is measured using the volume backscattering coefficient ($\mathrm{s_v}$), defined as:

$$s_v = \sum \sigma_{bs}/V_0, \tag{2.3}$$

where a sum over all discrete targets returning echoes in the sampled volume (V_0) is taken. There is a linear relationship between the abundance of fish and $\mathrm{s_v}$ as long as the species or group of species is known. For more details on $\mathrm{s_v}$, see Simmonds and MacLennan [52]

Furthermore, it is important to exclude the bottom echo when fish are being surveyed. As some fish like the sandeel may be found in schools close to the bottom, and if the discrimination of the bottom is poor, there will be large errors in the estimated fish density. For more details on bottom detection and implications, see Simmonds and MacLennan [52].

2.2 Machine Learning

Machine learning can be split into four parts; the *algorithm*, *empirical data*, a *task*, and a *performance* measure. A machine learning algorithm is designed to increase performance on a task, given data. During this process, also called *training*, the algorithm is said to be *learning* by fitting a *model* to the data. The two machine learning approaches used in this thesis are supervised learning and unsupervised [12].

2.2.1 Algorithm Approaches

2.2.2 Supervised Learning

Supervised learning algorithms are defined by the data consisting of an input and the desired output [12]. The algorithm will have to learn a function, mapping from input to correct output. In classification problems, the output would be a class label, for example, classifying pictures of cats from other animals. While in regression problems, the output is a value within a numerical range. For example, predicting the height of a person.

2.2.3 Unsupervised Learning

Unlike supervised learning, unsupervised learning algorithms only receive the input and learn properties contained in the data [12]. A practical example is clustering, where the samples in a dataset are divided into clusters of similar properties.

2.2.4 Data and Features

The quality of the input data to a machine learning algorithm will likely affect its performance [40]. Data must be gathered, integrated, cleaned of errors, preprocessed, and features often extracted before being used in learning. Hence, time allocated to prepare and increase the data quality can exceed the time spent learning. The process of extracting features is often referred to as *feature engineering*. It constructs a representation of the data with the most important factors to solve the task. This is often domain specialized and usually requires human involvement. In the next section, artificial neural networks (ANN) are introduced, which is one avenue within machine learning that can automate the extraction of complicated feature representations during learning.

2.3 Artificial Neural Networks

This section introduces the basic components of an ANN and how these are combined to create a *deep learning* network/ model.

2.3.1 Perceptron

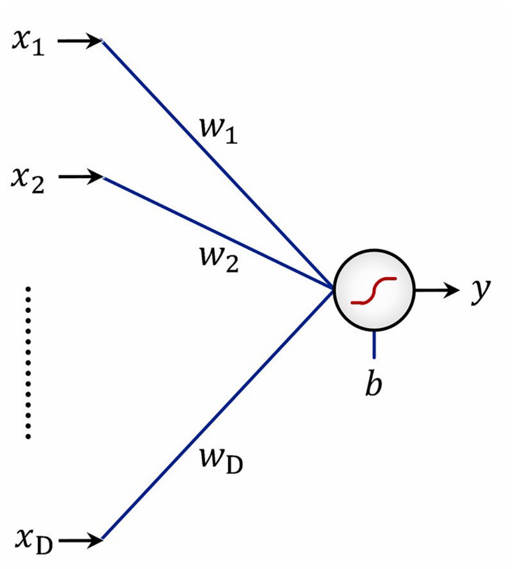
The ANNs fundamental building block is called an artificial *neuron* or *perceptron.* It consists of a linear regression with the tunable parameters w and b inside a non-linear activation function, explained later in section 2.3.3. The perceptron is formulated in the following way [47]:

$$y = g(\sum_{i=1}^{D} w_i x_i + b) \tag{2.4}$$

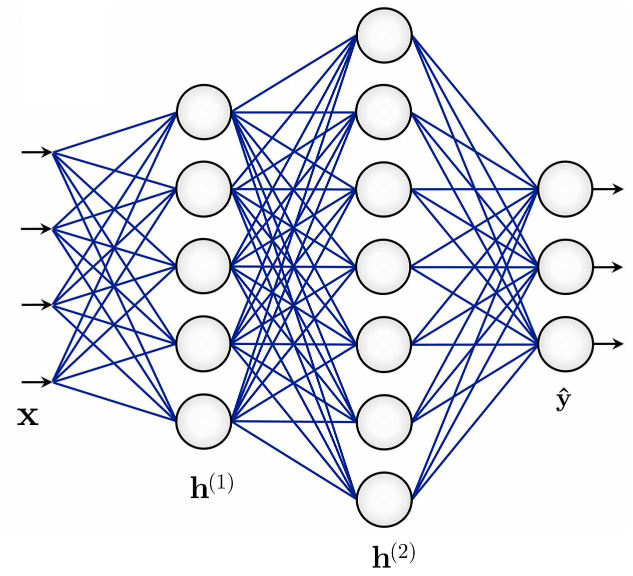
where D is the number dimension of the input space, x is the input vector, w is a set of weights of the same size as x, b is the bias, and g is the activation function. The single output value y, also called the neurons' *activation*, is a weighted sum of the input and weights plus the bias, transformed by the activation function. The perceptron is illustrated in figure 2.3.

2.3.2 Multi-Layered Perceptron

The neurons presented in section 2.3.1 are organized together in layers to form an ANN, which in turn forms what is called a multi-layer perceptron (MLP) [47]. If all neurons in each layer are connected to every neuron in the next layer, they form a type of ANN called *fully connected* networks. An MLP is depicted in figure 2.3.



(a) *Perceptron/neuron.*



(b) *An MLP with four inputs in the input layer(arrows to the left), two hidden layers($\boldsymbol{h}$), and three outputs in the output layer($\hat{\boldsymbol{y}}$).*

Figure 2.3: *Illustration of a single neuron and a fully connected deep learning network of neurons.*

Credit: Razavi [47]

The *architecture* of any ANN consists of an input layer, a user-defined number of *hidden layers*, and finally, an output layer [47]. More hidden layers form a *deeper* network, hence the name *deep learning.* An MLP is a type of network called a *feed-forward* ANN because the data flow from the input to the output layer, and each layer is a function of the previous layer. During training, the weights between every neuron and the bias are optimized in a process that is further explained in section 2.4. In the MLP depicted in figure 2.3, different neurons will activate with varying strengths depending on the input, resulting in different outputs. The architecture of the MLP in figure 2.3 can be expressed as [12]:

$$\mathbf{h}^{(1)} = g^{(1)}(\mathbf{W}^{(1)T}\mathbf{x} + \mathbf{b}^{(1)}) \tag{2.5}$$

$$\mathbf{h}^{(2)} = g^{(2)}(\mathbf{W}^{(2)T}\mathbf{h}^{(1)} + \mathbf{b}^{(2)}) \tag{2.6}$$

$$\hat{\mathbf{y}} = g^{(3)}(\mathbf{W}^{(3)T}\mathbf{h}^{(2)} + \mathbf{b}^{(3)}) \tag{2.7}$$

where for each layer $\mathbf{h}$ is a vector of activations, $\mathbf{W}$ is a matrix of weights, $\mathbf{b}$ is a vector of **biases**, g is an activation function applied element-wise, and $\hat{\mathbf{y}}$ is a vector of outputs.

2.3.3 Activation Functions

The activation function enables ANNs to learn non-linear features [47]. It is needed because a network consisting of only linear layers will be the same as a single linear layer. Hence, it won't be able to capture non-linearities in the data, and therefore an activation function is required in the hidden layers. Some activation functions can also be applied to the network's output to solve the task the network is set to perform and must be suited for the task at hand [12]. The activation function commonly used in the hidden layers is ReLU, which stands for *rectified linear unit* [51]. ReLU takes a real number as input and outputs this number if it is above zero, otherwise, it will output zero. Letting g denote the activation function, and x the input, the ReLU activation function can be formulated as follows:

$$g(x) = max(0, x) \tag{2.8}$$

ReLU is also visualized in figure 2.4. ReLU activate some neurons to propagate their input while preventing others from doing so. This can result in greater efficiency and faster training, as not all neurons are active, further detailed in Sharma [51].

Another activation function is the logistic sigmoid that transforms all input values to values in the range $[0 \ldots 1]$ [51]. It could be applied to the output layer to solve binary classification problems, as the values can be treated as probabilities. The formula for logistic sigmoid is:

$$g(x) = \frac{1}{1 + e^{-x}} \tag{2.9}$$

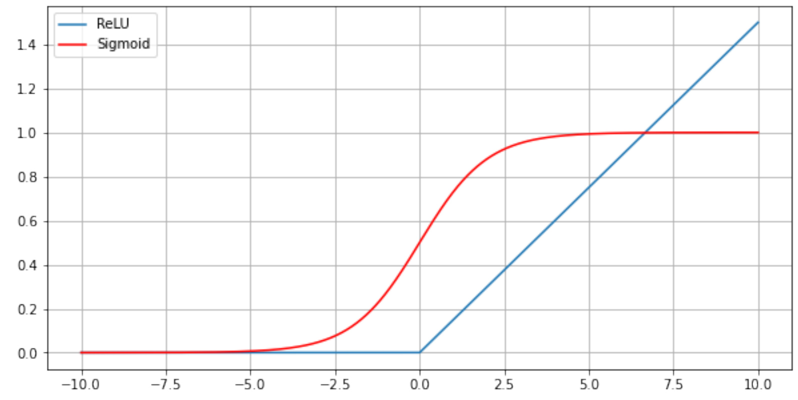


Figure 2.4: *A ReLU function (blue) and a sigmoid function (red)*

.

The Softmax Function

The *softmax* can be viewed as a multivariate version of the logistic sigmoid activation function, which allows the softmax to be applied to problems containing multiple classes [51]. For all data points, it calculates the probability of every class and can be expressed as:

$$g(\mathbf{x})_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}} \text{ for j = 1,...,K.} \tag{2.10}$$

where K is the number of classes, and the output summarizes to 1 over all classes. For a network solving multiclass classification, the output layer will have size equal to K. This corresponds to 3 classes in figure 2.3. The *softmax* would then be used as the last transformation ($g^{(3)}$ *in equation 2.5*) and output the probability of an input belonging to each of the three classes.

2.3.4 Convolutional Neural Network

The convolutional neural network (CNN) is a type of ANNs primarily used in machine learning tasks concerning images [43]. One of the reasons why the CNN emerged was because images input to a regular ANN produces a large number of learnable parameters. For example, a low-resolution image with 512×512 pixels passed to an hidden layer containing only one neuron would have $1 \times 512 \times 512 = 262144$ weights alone. To solve this issue and have fewer learnable parameters, the modern CNN is built around three main components; a *convolutional layer*, a *pooling layer*, and a *fully-connected layer*. An example CNN is illustrated in figure 2.5, and each main component will be explained later in this section.

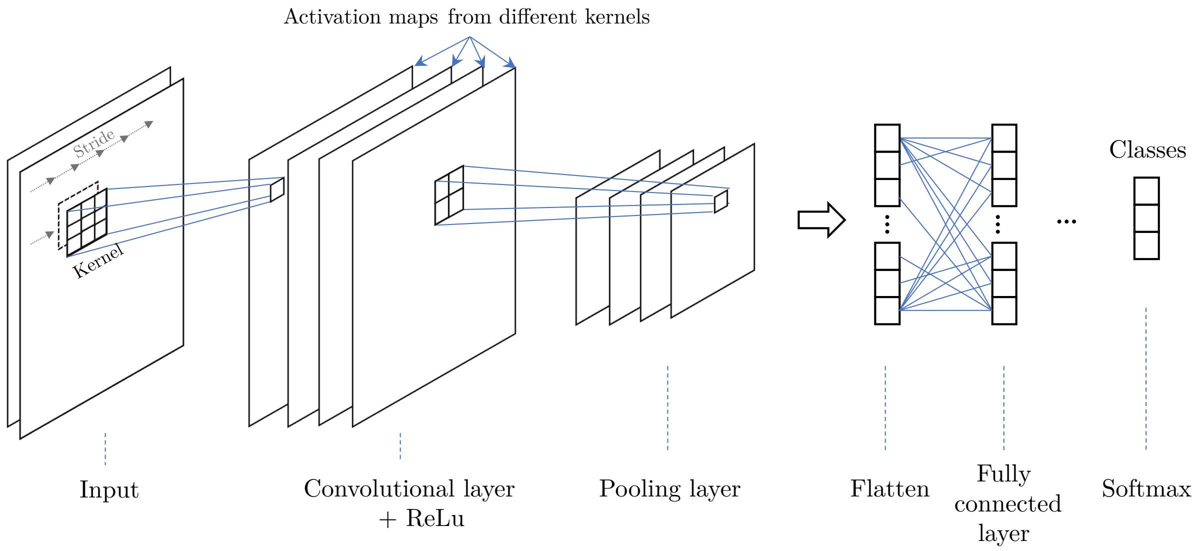


Figure 2.5: *Illustrations of the main components in a CNN.*

Convolutional Layer

O'Shea and Nash [43] describe the convolutional layer as consisting of many learnable multidimensional weight matrices that slide over the input. We will refer to such a matrix as a *kernel*. The kernel's height and width are parameters defined by the user, but the depth will always be equal to the number of channels in the input. This results in kernels being described only by *height* $\times$ *width*. The kernel slides over the input, and is applied to different locations of the input, also called the current *receptive field*. A single scalar value is computed when applied, which is the weighted sum of the kernel's weights and the corresponding values in the receptive field. If we have a 2-dimensional image $I(i,j)$ as input, a convolutional operation is expressed as [12]:

$$(I * K)(i,j) = \sum_{height} \sum_{width} I(i - height, j - width) K(height, width) \tag{2.11}$$

where $*$ is the convolutional operation, *I(i,j)* is the image pixel at *(i,j)*, and K is the kernel.

The output scalar value from the convolutional operation is usually fed through non-linear activation functions like ReLU and then called the *activation.* The sliding operation is based on a value called *stride*, which is the number of horizontal positions to move the kernel in the input between each calculation. Suppose we started from the left, and it is impossible to move horizontally to the right and still fit the kernel inside the input. In this case, the kernel will, if possible, move rows down vertically equal to the stride and then continue horizontally, starting afresh from the left. After sliding over the entire input, a complete 2-dimensional *activation map*, also called a *feature map*, has been created, one such map for each kernel applied. The idea is that each applied kernel will learn to identify different features in the input. An example of a horizontal edge detector can be seen in figure 2.6 [43].

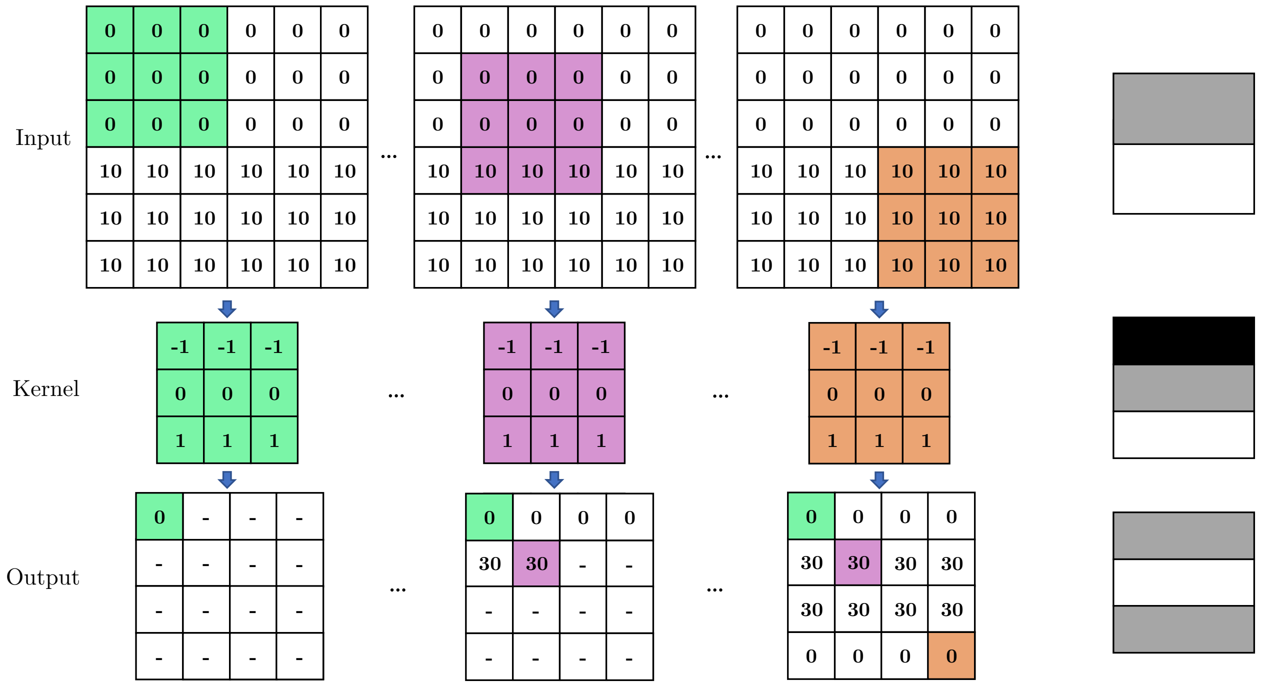


Figure 2.6: *Illustration of a valid (detailed later in this section) convolutional operation. The kernel is applied repeatedly across the input. The input size is 6×6, kernel size is 3×3, and stride 1, resulting in overlapping operations and output size being 4×4. The figures to the right show the input, kernel, and output(activation) as color gradings, where the color gets darker if the values are low. This example is a horizontal edge detector, and the result is large values in the activation map along the border between the values of 0 and 10 in the input, which could have been colors in a picture.*

The receptive field will start as small regions, but as we apply additional convolutional layers, it will have access to increasing context in regard to the input [12]. This is illustrated in figure 2.7, and kernels in early layers learn to identify simple features while later combining these to identify complex features. The kernels utilizes *parameter sharing*, as the same weights are repeatedly used across the input. Furthermore, the kernels are often smaller than the input, resulting in *sparse connections* as opposed to fully connected networks. The parameter sharing results in the CNN having another useful attribute called *equivariance*, which means that if the input changes, the output changes in the same way.

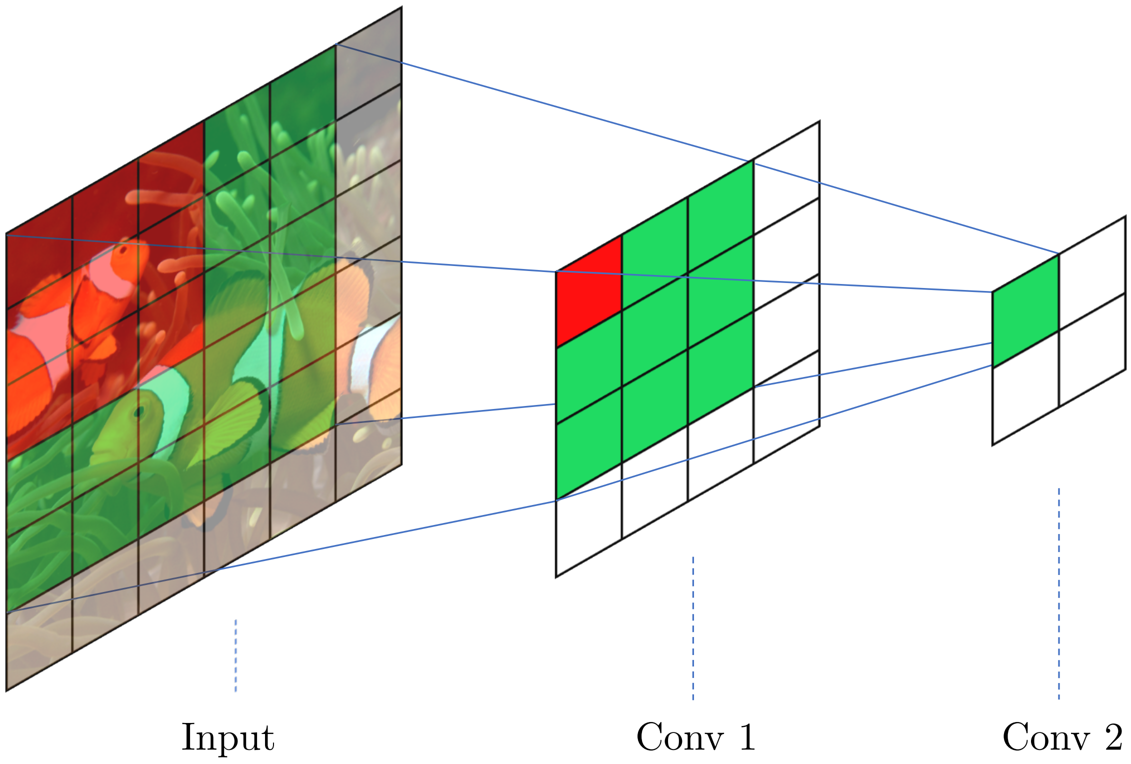


Figure 2.7: *The activation maps from two convolutional layers with 3×3 kernels and stride 1. The first convolution's receptive field is marked as red. On its activation map, a new convolutional layer is applied. Its first receptive field is outlined in green, which translates to a larger area in the input.*

Credit: Original image by Nick Hobgood [6], used as input picture above (Edited with colored grid).

The application of CNNs on acoustic data can be motivated by the echograms being similar to regular images, but the RGB color channels being replaced with the different frequency channels measured. However, the same object will be represented differently in echograms at varying ranges from the transducer, possibly similar to objects of another class [52]. This likely increases the complexity of target classification in acoustic data, as benefits of equivariance is not as applicable.

Reductions in the spatial size will normally occur with the convolutional operation described in this section, and such operations are called *valid convolutions* [43]. By applying padding with zeros around the input, we can retain the dimensions of the input. The effect is that more convolutional operations fit in the new padded input, hence an equal output size. This is called the *same convolution*, illustrated in figure 2.8.

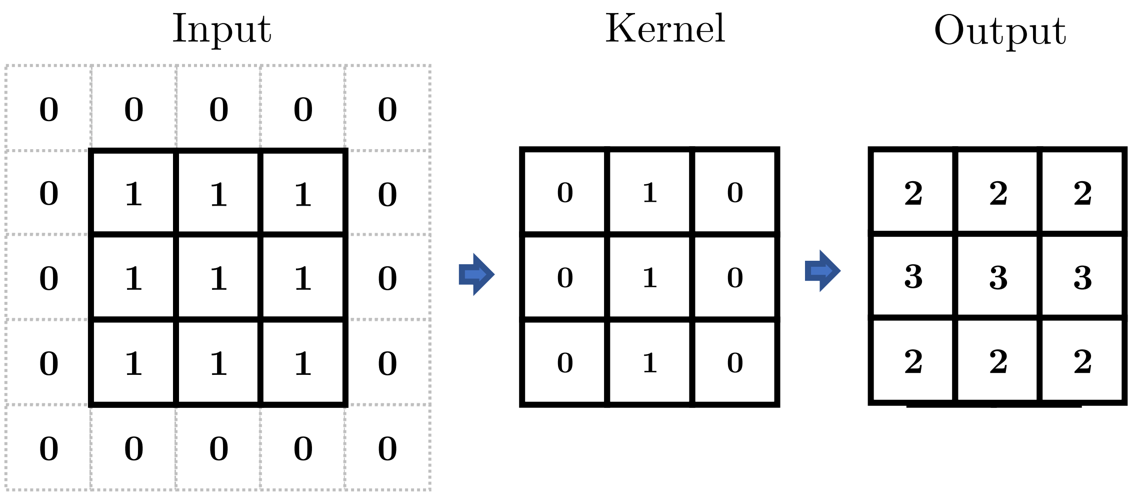


Figure 2.8: *Illustration of the same convolutional operation. The input size is 3×3, but after padding with zeros, the size is 5×5, kernel size is 3×3, and stride is 1. This results in an activation map size of 3×3, conserving the input size.*

The Max Pool Layer

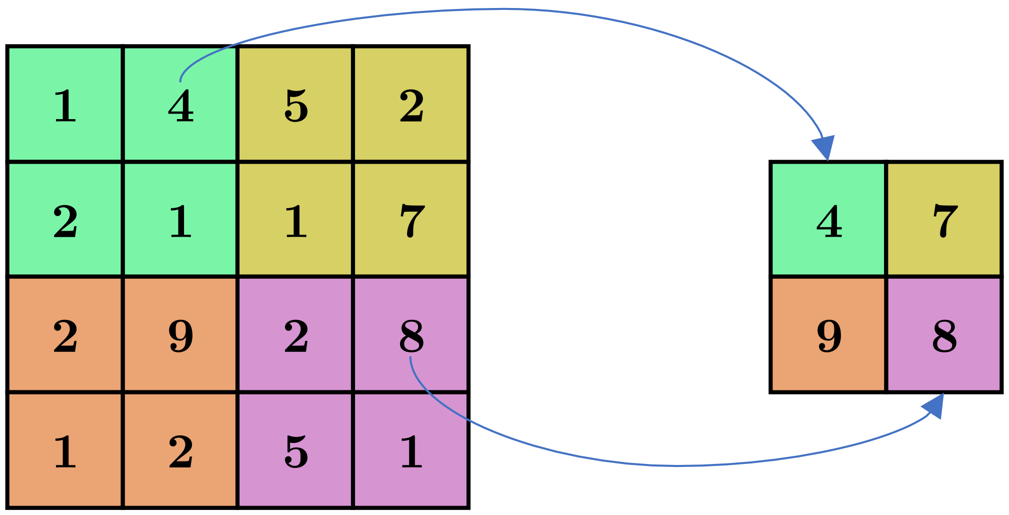


Figure 2.9: *Illustrates the max pool operation with size 2×2 and stride 2.*

The max pool layer reduces the *height* and *width* of its input [43]. Like a convolutional operation, the max pool looks at an input region but instead applies a *max* operation. The pooling kernel size is given in *height* × *width* and is applied individually to each channel of the input. This reduces the height and width but preserves the number of channels. The most common max pool layer is a 2×2 with a stride 2. Alone, the max pool has no learnable parameters and is applied to decrease the computational complexity of the CNN.

Fully Connected Layer

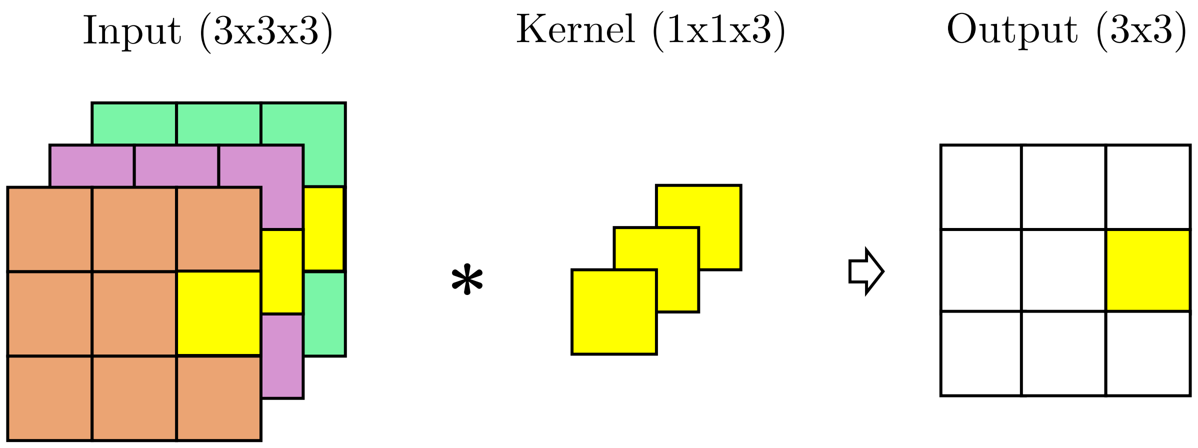


Figure 2.10: *Illustration of the 1×1 convolution with stride 1. Yellow cells demonstrate the application of a 1×1 kernel along all channels of the input, producing one activation in the output.*

Lin et al. [29] proposed the convolutional layer with kernel size 1×1 and stride 1, followed by an activation function. The 1×1 layer will take the weighted sum along a 1×1 slice through all channels of the input, as illustrated in figure 2.10. This is equivalent to applying a fully connected layer to the same values. As this preserves the *resolution*, it can be used to alter the depth of the output feature maps by specifying the desired number of kernels, while also introducing non-linearity. In figure 2.10, we have only one kernel, but if two were applied instead, the output feature map would have a depth of two. In this work, it is used mainly to map high dimensional feature maps to the desired number of classes.

Transposed Convolutions

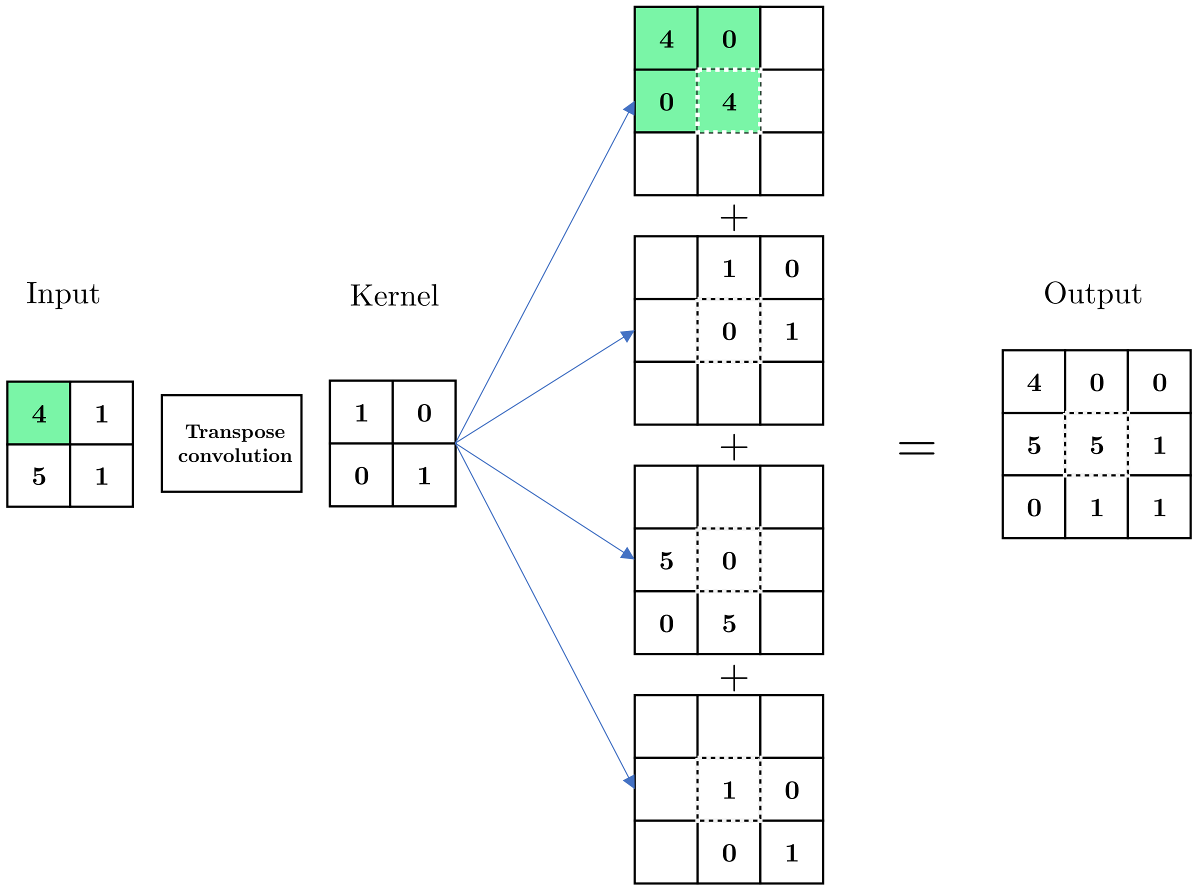


Figure 2.11: *Illustration of the transposed convolution operation with kernel size 2×2 and stride 1. The green color shows one of the intermediate computations. The center value of each crop is outlined to illustrate the summation step as these overlap.*

A transposed convolution is an operation taking an input, and with a kernel similar to that described in 2.3.4, but now instead map the input to a higher resolution [9]. In example figure 2.11, a 2-dimensional 2×2 input is fed to a transposed convolutional layer with kernel size 2×2. The whole kernel is multiplied element-wise with the input and proceeds to produce values in a temporary matrix initialized with zeros, denoted by empty cells in the figure. We do not use the temporary values in practice, but they are used here to illustrate the intermediate computations. The calculated values in the temporary matrix are situated correctly relative to the input. These temporary matrices are then summed over, producing the output. This operation is repeated for all channels, retaining the depth of the input.

Segmentation

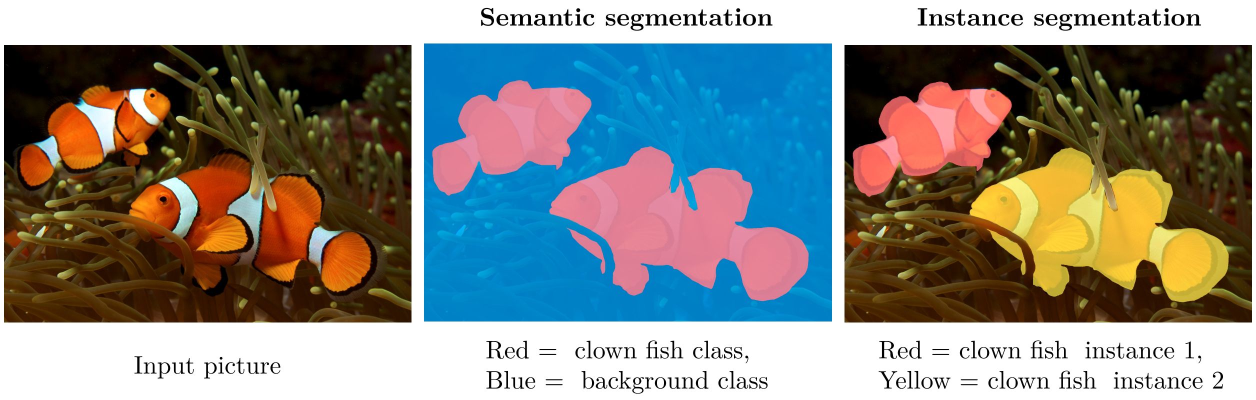


Figure 2.12: *Illustration of the difference between semantic and instance segmentation.*

Credit: Original image (*Input picture above*) by Nick Hobgood[6]

Segmentation is a task where the objective is to assign one or several classification masks to the input, usually a picture [14]. This is further split into two different categories: *semantic* and *instance* segmentation. In semantic segmentation, we assign each pixel in the input to predefined classes. The output would have the same resolution as the input but with channels equal to the number of classes. A softmax would then be calculated for each pixel across the depth, and the pixel would be assigned to the class with the highest probability. Hence, producing a mask for each class. In instance segmentation, we increase the complexity by applying semantic segmentation while simultaneously assigning a bounding box to each object, as visualized in figure 2.12.

2.4 Training Neural Networks

This section will describe the main concepts behind how neural networks are trained to perform on various tasks.

2.4.1 Forward-Propagation and The Loss Function

The objective of an ANN is to approximate some optimal function f, and in this thesis, we focus on classifiers, $\mathbf{y} = f(\mathbf{x})$, which maps an input $\mathbf{x}$ to an output category $\mathbf{y}$. The ANN approximates this function by defining a mapping, $\mathbf{\hat{y}} = \hat{f}(\mathbf{x}, \theta)$, and learns the values of the parameters θ (*weights and biases*) through training using examples. In supervised tasks, the labels instruct the output layer exactly how to perform given the input data. However, the data does not inform the individual *hidden layers* how to behave to produce this desired output. When the data flow through the network using the parameters θ, it produces outputs $\mathbf{\hat{y}}$, called the *forward-propagation step.* How the parameters are initialized can heavily affect the training process, and different strategies are further described in Goodfellow et al. [12]. Using a *loss function* to compare the true $\mathbf{y}$ values to the estimated values $\mathbf{\hat{y}}$, we measure the network's *error*, also called *loss.* The network uses this loss to then alter θ to best approximate *f*, which will be explained later in section 2.4.3. In classification tasks, the network is trained to output the probability of each class given an input [15]. We can use a loss function called *weighted cross entropy* to train such a model. This function outputs a loss based on the probabilities, weights classification of certain classes differently, and is often used when dealing with data containing class imbalance as more weight can be applied to the minority class. Expressed as:

$$loss(x, y) = -\sum_{k=1}^{K} w_k y_k \log(\hat{y}), \quad \hat{y} = \hat{f}(x, \theta) \tag{2.12}$$

where K is the number of classes, w_k is weight for class k, and y_k is the target label. More examples of loss functions can be viewed in Mishra and Gupta [37].

2.4.2 Mini-batch Stochastic Gradient Descent

> "*A recurring problem in machine learning is that large training sets are necessary for good generalization, but large training sets are also more computationally expensive."* - (Goodfellow et al. [12] 2016, page 147)

Calculating the total loss of the whole dataset is often unfeasible, and depending on the hardware, this could lead to a crash or slow learning due to heavy memory demands. A solution is to sample a set of examples, called a *mini-batch*, from the entire dataset, with the intent to approximately estimate the true loss using this smaller fraction of the dataset.

Then we update the parameters of our network based on this and repeat on a new mini-batch. This is called mini-batch *stochastic gradient descent (SGD)*, a common *optimization* algorithm. The size of this mini-batch can vary from one example to hundreds. When we have run this process on all the data, we say that an *epoch* has passed [12].

2.4.3 Back-Propagation and Gradient-based Learning

To update the network parameters, we use the loss from a mini-batch and iteratively step back through the layers in a process called *back-propagation* [49]. In each step, we calculate the *gradient* of the loss functions with respect to the parameters of the current layer by using the chain rule. This is to determine how changes to each parameter will affect the *loss*. Using the gradient, SGD performs *gradient descent* [12] by updating all parameters in the opposite direction of the gradient to reduce the loss. In what magnitude a parameter is adjusted by the optimizing algorithm is determined by the *learning rate*, usually a value between 0 and 1. The parameter update is expressed as [19]:

$$\theta^{(j)} \leftarrow \theta^{(j)} - \eta \frac{1}{m} \sum_{i=1}^{m} \frac{\partial loss(x_i, y_i)}{\partial \theta^{(j)}} \tag{2.13}$$

where θ is the parameters of the network, m is the mini-batch size, η is the learning rate, and j is the layer. In figure 2.13 an example loss function is illustrated with one global *loss* minima and different learning rates applied with SGD. Low learning rate values usually have a long training time and may cause the SGD to converge to a local minima instead of the global [10]. However, too high values can overshoot the global minima and diverge. Both can be prevented by applying a method to adapt the learning rate to the *topography* of the loss function. This thesis applied *momentum*, which only acts as a *velocity* to the update step. The *velocity* is based on past steps, and the update will step in the *velocities'* direction, not the current *gradient*. More detail on *momentum* can be found in Sutskever et al. [53].

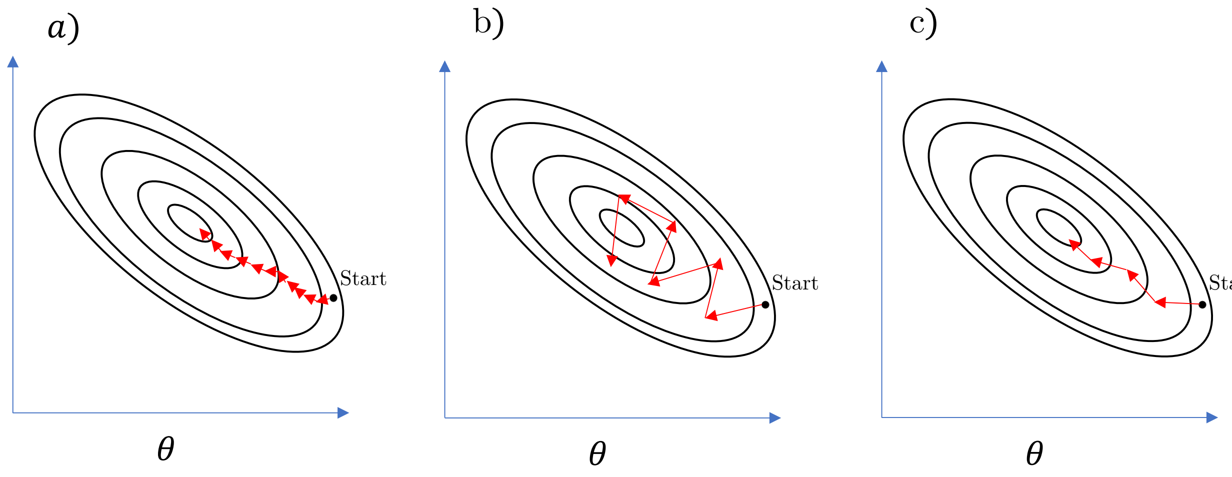


Figure 2.13: *Three different applications of SGD on a loss function. Each arrow is an imagined learning step taken by the algorithm for; (a) low learning rate, (b) high learning rate, and (c) momentum.*

In summary, the entire training process using SGD can be described as the following algorithm [10]:

```
Mini-batch SGD one epoch
Loop:
1.  Sample a mini-batch of data.
2.  Forward propagate the mini-batch through the network and compute the loss.
3.  Back propagate to calculate the gradients.
4.  Update the parameters based on the gradients.
```

2.5 Model Evaluation

To evaluate a machine learning algorithm, we need a performance measure. First, the *performance metric* itself will be described, followed by a technique applied to make unbiased measures of the model by leaving out parts of the data. Finally, two central challenges that appear in machine learning.

2.5.1 Performance Metrics

This section describes the performance metrics used in this thesis. Consider a binary classification system that classifies samples into either the *positive* or *negative* class. Predictions by the classifier can thus be sorted into the following four categories [45]:

- **True positive (TP):** A correct classification of a positive example.
- **True negative (TN):** A correct classification of a negative example.
- **False positive (FP):** A negative example incorrectly classified as positive
- **False negative (FN):** A positive example incorrectly classified as negative.

We can now calculate the performance of the classifier from these values, and the simplest is accuracy [45]:

$$accuracy = \frac{\textit{correct predictions}}{\textit{total number of predictions}} = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.14}$$

This metric does not handle class imbalance well, as it is equivalent to calculating the percentage of correct predictions [45]. An example is that if 95% of the data belongs to one class, then always predicting this class will give us an accuracy of 95%.

We calculate two new metrics to better deal with class imbalance: *precision* and *recall* [45]. Precision is the percentage of positive predictions made by the model that are correct. Recall is the percentage of all positive samples the model managed to classify correctly.

$$precision = \frac{TP}{TP + FP} \tag{2.15}$$

$$recall = \frac{TP}{TP + FN} \tag{2.16}$$

Then, by using precision and recall, we calculate the *F1-score* [45]. It combines these metrics and is designed to work well on imbalanced data. The F1-score formula:

$$\textit{F1-score} = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{2.17}$$

2.5.2 Train-Validation-Test Split

When a machine learning model is learning, the goal is to achieve the lowest *generalization error*. This means to not only perform well on data seen during training, but also on new unseen data. To measure this error, it is normal to split our data into three parts: the *training*, *validation*, and *test* datasets, and we measure some error or metric on each. As the name suggests, the training data is used during the training process of the model. The validation dataset is extracted from the training dataset and gives an unbiased estimate of the models' performance and can be used to guide the training process. The last mentioned could be to select the best model from a selection of many. Finally, the test dataset is used to get an unbiased estimate of the final model's *generalization error* [12].

2.5.3 Overfitting Vs. Underfitting

A model's performance depends on the difference between its *training* and *test* error. *Underfitting* happens when a model fails to achieve a low training error, while *overfitting* happens when the training error is significantly lower than the test error. To manipulate this behavior, we adjust the model's *capacity.* Capacity represents the variety of functions the model can learn, and by adjusting it, one can increase and decrease the likelihood of underfitting and overfitting. The capacity can be controlled by, for example, changing the number of layers in a neural network, and further details can be seen in Goodfellow et al. [12]. Low *capacity* means that the model may fail to capture patterns in the data. High *capacity* translates to adjusting to the training data to such an extent that the model performance may be worse when given unseen test data. The optimal solution, depicted in the center plot of figure 2.14, is to have a model with a balanced capacity that is as close to the true function as possible [12].

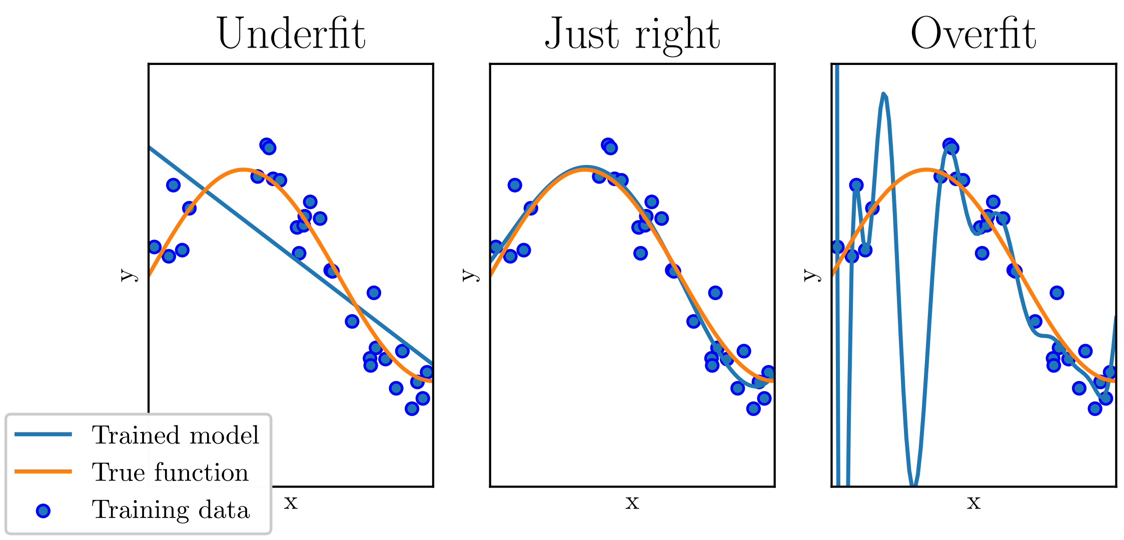


Figure 2.14: *Training data is generated with random noise around a sinus wave (True function). Model capacity increases from left to right. The center plot illustrates a model that has learned an almost perfect fit to the true function.*

2.6 Regularization

Regularization, as described by Kukačka et al. [28], is any supplemental technique with the goal of increasing the model's generalization performance. Two techniques used in this thesis will be described here; *batch normalization* and *data-augmentation*.

2.6.1 Batch-Norm

Batch normalization is a technique applied to reduce what is called *internal covariate shift* [19]. This is defined as the change in the distribution of activations in hidden layers caused by the change in the network's parameters when training. During *backpropagation*, the hidden layers depend on the activations of all layers before them. As each layer changes its output distribution, the other layers must adapt to this change. Research has shown that this slows down and destabilizes the training process, and a solution to this problem is the implementation of *batch normalization*. Using the activations from all the neurons in a hidden layer, a *mean* and *variance* are calculated per mini-batch. These values are then used to normalize the activations of the hidden layer. Each hidden layer is given two additional learnable parameters γ and β, that perform a linear transformation of the normalized activations, defined as such:

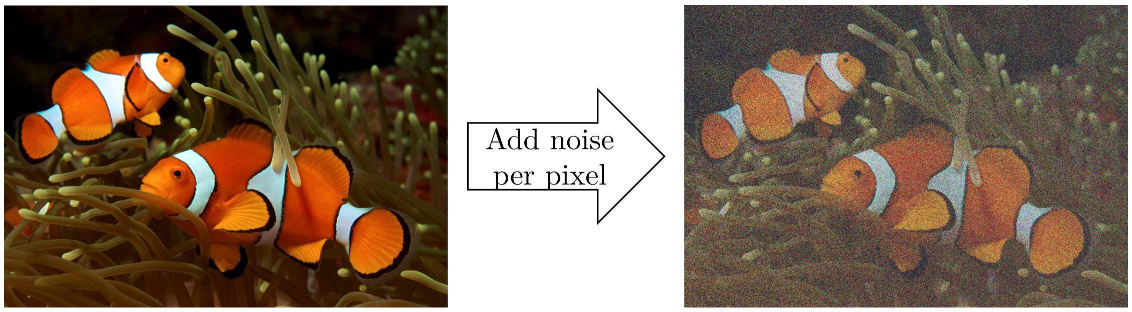
$$\hat{Z}^{(i)} = \gamma Z^{(i)}_{norm} + \beta \tag{2.18}$$

where $\hat{Z}^{(i)}$ is the batch normalized activations, $Z^{(i)}_{norm}$ is the normalized activations for the i^{th} hidden layer. The learnable parameters make the ANN able to adjust and shift the distribution through the training process. The result may lead to a faster and more stable training process.

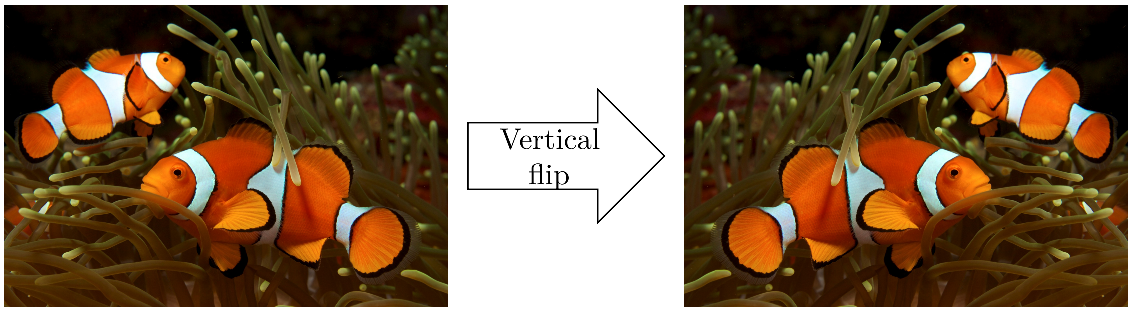
Recently, the poor understanding of batch normalization has come into question, and Santurkar et al. [50] have stressed that more investigation should be put into understanding its effectiveness. Their findings show that it might not stem from internal covariate shift but likely other factors.

2.6.2 Data-Augmentation

Data-augmentation is a regularization method directly applied to the training dataset by applying some transformation [28]. Several methods are available, but the two used in this work are: *adding noise* and *flipping*. One example of applying noise is to add Gaussian values with a mean of 0 and some user-defined variance to each pixel. This adds more randomness to the data, making the model learn more general features instead of specific. The network is less prone to overfit on certain samples, which in turn might increase generalization performance. Other methods of applying noise are described in Kukačka et al. [28]. Flipping is a simple transformation where we flip the input and label along a particular axis. Thus, mapping the data to a new representation. An example of each method mentioned can be seen in figure 2.15:



(a) *Gaussian noise is added to each pixel.*



(b) *Vertical flipping.*

Figure 2.15: *Two augmentation methods applied to the same image.*

Credit: Original image (*Both left pictures above*) by Nick Hobgood [6]

2.7 U-Net

In this section, we introduce the architecture of the model that is the backbone of the work in this thesis. U-Net is a fully convolutional, state-of-the-art [46] semantic segmentation CNN initially developed for biomedical image analysis by Ronneberger et al. [48].

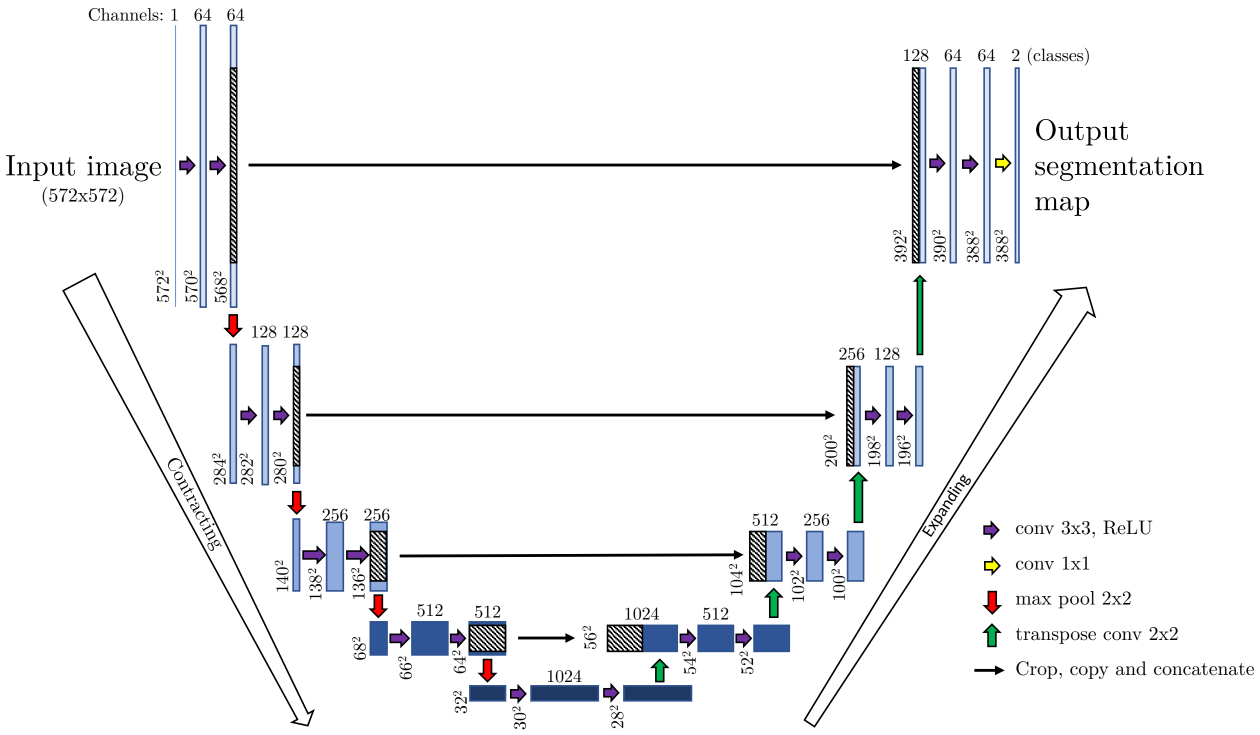


Figure 2.16: *The U-Net architecture, the downwards facing arrow illustrates the contracting path and the one facing upwards is the expanding path. For each block, the vertical number is the resolution, while the horizontal is the number of feature channels. The color gets darker as the channels increase.*

Credit: Ronneberger et al. [48]

U-Net utilized what Ronneberger et al. [48] called a *contracting path* to identify what was in a picture, while an *expanding path* localized where it was. These two branches were more or less symmetrical, and together they formed a U-shape, giving the network its name. The contracting path can be looked at as five different stages of processing, from top to bottom, in figure 2.16. Each stage consisted of two 3×3 valid convolutions with their individual ReLU activation functions. Initially, the feature channels are increased to 64, and the channels were doubled for each contracting stage. The convolutions were followed by a 2×2 max pooling operation with stride 2 to further decrease the resolution of the output from the convolutional operations and then output a feature map to the next stage. After the bottom stage, the max pool operation is replaced with transpose convolutions to now increase the resolution. At each subsequent stage traveling back up the expanding path, the number of feature channels is halved by the convolutional steps down to 64.

In the expanding path, the previous stage's output was concatenated with a crop from the output feature map of a stage from the contracting path with the same channel size, the cropping is due to different resolutions. This step allows the following expanding path convolutional operation to access both the high-resolution feature map from the contracting path and the upsampled feature map, which in combination helps with the localization of features [48].

At the final layer in the expanding path, a 1×1 convolution maps the 64 feature channels to the user defined number of classes, which is two classes in figure 2.16. Finally, the softmax was then calculated between these classes, giving each pixel a probability distribution over the classes, with one channel for each class. Hence, giving us a segmentation map [48].

When released, U-Net outperformed other networks in multiple biomedical challenges [48]. Its performance inspired new models that use the U-Net architecture as their backbone, as seen in NAS-Unet [56] in 2019, and Unet++ [58] in 2018. U-Net has also been applied successfully to other fields such as road extraction in satellite images by Zhang et al. [57] in 2018, and on acoustic classification by Brautaset et al. [3] in 2020, which will be explained in the following chapter.

2.8 Knowledge Distillation

Knowledge distillation (KD) is a method within deep learning which focuses on transferring the knowledge from one *teacher* network with strong capability to a *student* network. One example of KD is using a pretrained teacher network with high performance to label the data, and then we train the student using these labels[1]. Alternatives have been to extract knowledge from the layers of the teacher and use this during training of the student with or without teacher labels. Generally, the methods are applied in three principal situations [1]:

- Create a new, less complex model for platforms with computation power limitations.
- Enhance the accuracy of an existing model.
- Train a model with limited or constrained data.

In some instances, KD has been applied to train student models even more complex than the teacher network, with some methods using an ensemble of teacher models [1]. The research field of KD has attracted much attention in recent years but is still under development. This causes applications of KD not to follow a strict set of rules but is a creative process adapted to each domain applied. However, recent studies have shown promising results for KD, as detailed in Alkhulaifi et al. [2].

[1]The teachers labels can either be binary classes values called *hard* labels or the class probability distribution from the softmax function called *soft* labels [11].

Chapter 3

Basis and Related Work

In this section, we introduce applications of deep learning on acoustic data. First, the work this thesis uses as a basis, then we describe some recent developments.

3.1 Acoustic Classification in Multifrequency Echosounder Data using Deep Convolutional Neural Networks

Brautaset et al. [3] had as objective to propose a deep learning method to classify and segment multi-frequency acoustic data gathered during acoustic trawl surveys, without using predefined features. Their work is the basis for the work contained in this thesis. Their architecture is visualized in figure 3.1, and the difference from the original U-Net implementation is the use of the *same convolutions*, reduction in input resolution to 256×256, and the use of *batch normalization.* Furthermore, as the resolution of the layers in the contracting and expanding path has the same size, hence when concatenating, cropping is no longer needed.

The Data

The data used originated from the ongoing Norwegian acoustic trawl surveys, where they used the data spanning 2007-2018. 2011-2016 was set as training and validation data, and

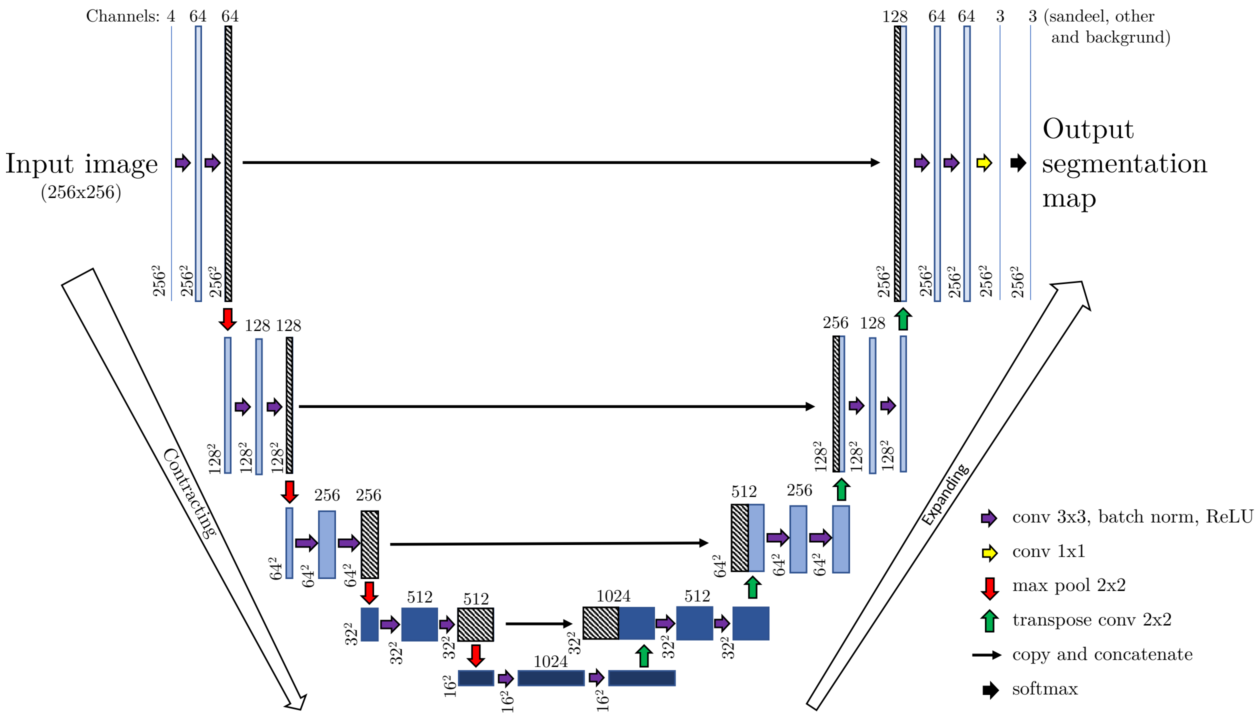


Figure 3.1: *Modified version of the original U-Net architecture (illustrated in figure 2.16) made by Brautaset et al. [3].*

Credit: Brautaset et al. [3]

2007-2010 combined with 2017-2018 as test data. For all years, the LSSS system was used to annotate the data. The frequency channels extracted from each year were s_v echograms at 18kHz, 38kHz, 70kHz, and 200kHz. Operator annotations initially contained the classes *sandeel*, *other*, *0-group sandeel*, and *possible sandeel*, and were annotated by the same operator. 0-group sandeel and possible sandeel were added to a new class *ignore*, which were ignored during training. Both of them were edge cases originating from an extraordinary event seen during a trawl or operator uncertainty. The annotations were converted to a pixel map of the same size as the s_v data, and all pixels not allocated to a class were set as the *background class*. Annotations were originally designed to summarize the s_v values over an area to estimate quantity. Hence, they were usually square and larger than the actual school of fish. This is not suitable as labels for a CNN as pixels around the edges in the annotations sometimes belonged to the wrong class. As a result, the annotations were approximately reshaped to be more similar to real schools of fish. The training methods applied is described in the methodology chapter (*chapter 4*), as it is heavily based on this work. In short, they trained the model on extracted crops from the echograms, with a focus on class balance regarding the information contained in the annotations of the crops [3].

Performance

They measured performance using two different methods [3]. The first method was to measure the performance over *entire echograms* and all its pixels. While the second method extracted predictions in *small regions* around and including existing annotations, and was applied due to the observations of many schools missing their annotations. The small regions would likely result in the decrease of many false positives being produced by the model, and would likely better reflect its actual performance. To have a benchmark to compare to, they used the traditional pipeline developed by Korneliussen et al. [27], which uses an acoustic feature library to identify species, that does not use machine learning methods. The metric used was F1-score, where *sandeel* was the *positive* while *other* and *background* was the *negative* [3].

Small regions resulted in an overall F1-score of 0.87 for their model on all the test data after a threshold of 0.8 was applied to the probabilities, while the benchmark achieved a F1-score of 0.77. When applying the model to entire echograms, the performance for 2017 and 2018 was deemed as satisfactory (F1-score 0.61 and 0.78 respectively), but worse on the earliest years ranging from 2007-2010 (F1-score 0.11,0.51, 0.78, and 0.68 respectively). The benchmark method also mirrored this, which achieved 0.03-0.62 over the same years. The decrease in performance over entire echograms was attributed to missing annotations and many unidentified features in the data being allocated to the *background* class. Data quality varied between different years due to changing weather conditions, fish populations, and software development stages used during annotations. The model was concluded as able to classify sandeel in acoustic multifrequency measurements reliably [3].

3.2 Related Work

In 2018 Korneliussen [25] reported the most recent methods used for acoustic target classification. The mentioned methods applying ANNs were most frequently applied to data containing a single frequency channel. Due to these being used in a supervised setting, they stressed the importance of high-quality annotations. Similar to the work detailed earlier in Brautaset et al. [3], we will describe some newer research applying deep learning to acoustic data.

Marques et al. [34] compared different machine learning methods for automatically interpreting multi-frequency echograms and proposed that a deep learning based end-to-end framework was best suited for the task. Their experiments handled the generation of square bounding boxes around schools of fish, and their results were comparable to or better than those produced by human operators. The work was later extended with instance segmentation capabilities in 2021 [33]. This produced more accurate predictions with pixel precision, and they described this as more befitting biological analysis.

In 2021 Choi et al. [5] proposed a semi-supervised[1] deep learning network to solve the reliance supervised ANNs have on the correct manual annotation of the acoustic data. Using the same data as Brautaset et al. [3], they utilized *two* loss functions which optimize the same CNN in alternating order. The first loss function seeks to *cluster* the data by finding intrinsic characteristics, and the second loss function uses classification based on the existing annotated data. Their model could outperform other traditional machine learning algorithms, both with or without missing annotations. Showing that we can efficiently process acoustic data with only a few annotated samples.

[1]A machine learning approach where you utilize both labeled and unlabeled data [12].

Chapter 4

Material and Methods

This chapter describes the steps taken in this thesis to answer the research question:

> "***As lightweight unmanned vessels may not have the capacity to carry all the six echo sounders the IMR usually deploys on the Norwegian sandeel surveys, which ones should be prioritized with the regard to classifying sandeel in multi-frequency acoustic data?***"

The thesis follows the work of Brautaset et al. [3], which was outlined in section 3.1 and will be referenced throughout this chapter. In summary, this chapter will first look at the data itself and the tools and methods applied to prepare it for the training of machine learning models, followed by a description of the experiment. The experiment focuses on training individual models with different subsets of frequencies and measuring the performance of each.

4.1 The Data

The data used in this thesis stem from the annual Norwegian acoustic trawl surveys from 2018 and 2019 using the Simrad EK60 echosounder, where 2018 was the same year used by Brautaset et al. [3]. The data from each year consisted of a collection of *.raw* files. The *.raw* files are the uncompressed raw output from the echo sounder, and stored the backscatter as s_v values in echograms for six frequencies *18kHz*, *38kHz*, *70kHz*, *120kHz*, *200kHz*, and *333kHz*. Two frequencies (*70kHz and 333kHz*) more than used by Brautaset et al. [3] . The settings on the echosounder resulted in the size of each pixel representing 1 second horizontally and 19.2 centimeters vertically [4] (*Visualized in figure 4.2*). The height and width of the echo sounder data were dependent on the depth measured and the total navigation time of the survey, and the two years used as data in this thesis contained 688 hours of data from 2018 and 1107 hours from 2019.

4.2 The CRIMAC-Pipeline Modules

Based on the work performed by Brautaset et al. [3], a pipeline for classifying the acoustic backscatter was created under IMRs project Center for Research Based Innovation in Marine Acoustic Abundance Estimation and Backscatter Classification (CRIMAC) and could process the .raw data into a format able to be used by Python [41]. The pipeline could be run as one whole module to get the predictions from the .raw data directly, or submodules of the pipeline (*illustrated in figure 4.1*) could be accessed and run separately. Most of the outputs from modules in the pipeline are of the *.zarr* format. This format stems from the Python Zarr package, which facilitates NumPy array loading and operations through *xarray* on data that is too large to be loaded completely in computer memory (*All tools used are described in appendix A*). This section will describe the function of each submodule, hereby called *module*, used in this thesis:

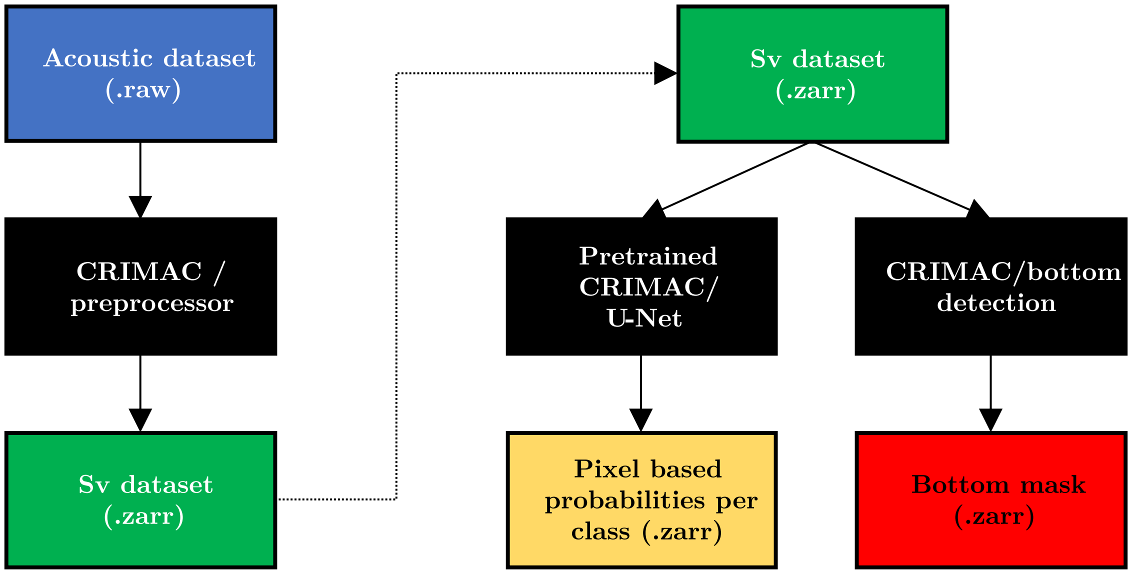


Figure 4.1: *Module overview and output flowchart. Black represents modules, and other colors represents input/output. The modules' and input/outputs' colors will stay the same for later illustrations.*

- **CRIMAC/Preprocessor:** Used to process the *.raw* files to the *.zarr* format, and re-grids all frequency channels to a common range if not equal. The output is a s_v dataset represented as a single multidimensional array of size $frequency \times ping \times range$ (*One echogram per frequency*). The output s_v file is illustrated in figure 4.2.
- **Pretrained CRIMAC/U-Net:** Using a pretrained U-Net model, it produces a segmentation map of pixel-based probabilities with spatial size $class \times ping \times range$. Output classes were *sandeel*, *other*, and *background*. The output predictions is illustrated in figure 4.2.
- **CRIMAC/Bottom detection:** Identifies the bottom and generates a binary pixel-based map stored as *.zarr*. This is a 2-dimensional array of size $ping \times range$. The output is shown in figure 4.2.

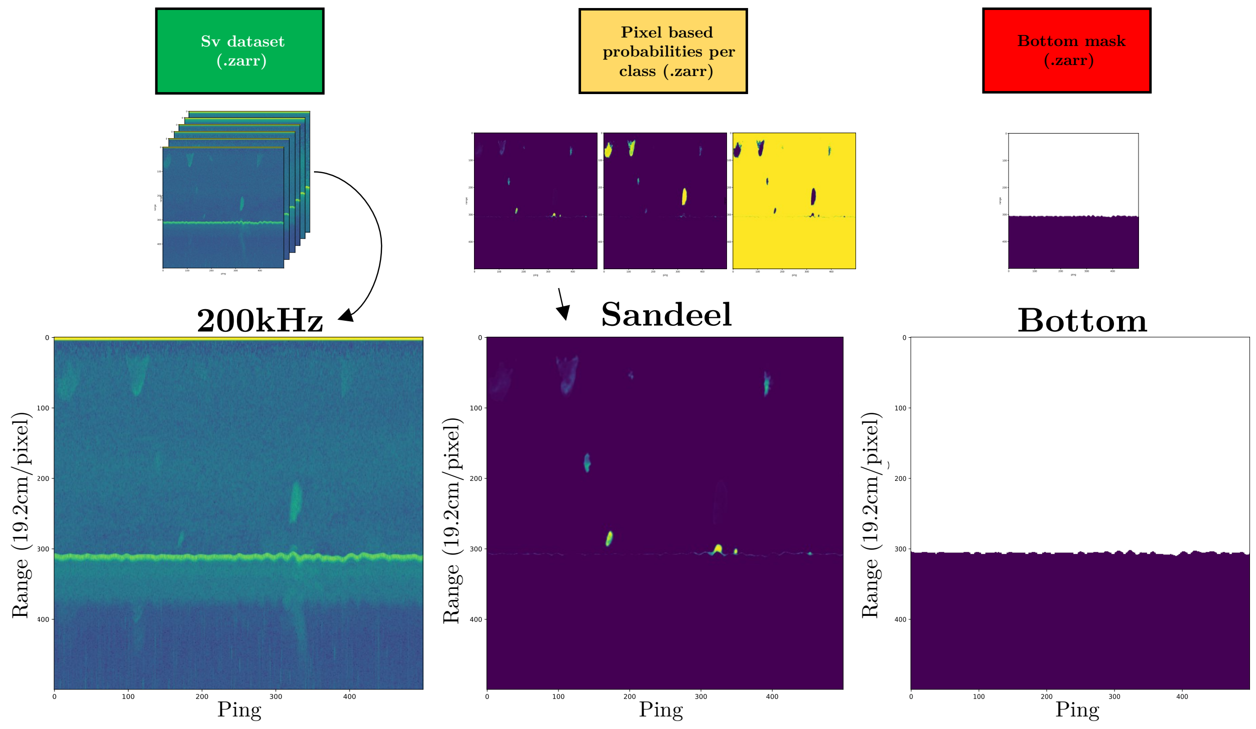


Figure 4.2: *Illustrations of the output from the different modules with one enlarged example from each output. This example is a* 500 × 500 *crop, while the real data contained millions of pings. The 200kHz has been transformed to the decibel scale to make the data observable. The color scale is set to be purple at 0 and yellow at 1. White denotes missing values.*

4.3 Pseudo Labels

The operator annotations were unavailable during this work, and so the pretrained CRIMAC/U-Net model was treated as a teacher model, from now called *baseline model*, and it's output as annotations. A hard threshold of 0.8 was applied to the baseline model's output predictions, and all values below were set to 0, and above to 1, resulting in a hard mask for each class, hereby called the *pseudo labels* (*visualized in figure 4.3*). This threshold was applied to only include the most certain predictions of the pretrained CRIMAC/U-Net. As this threshold was applied to all classes, there were instances where a pixel was not assigned to a class.

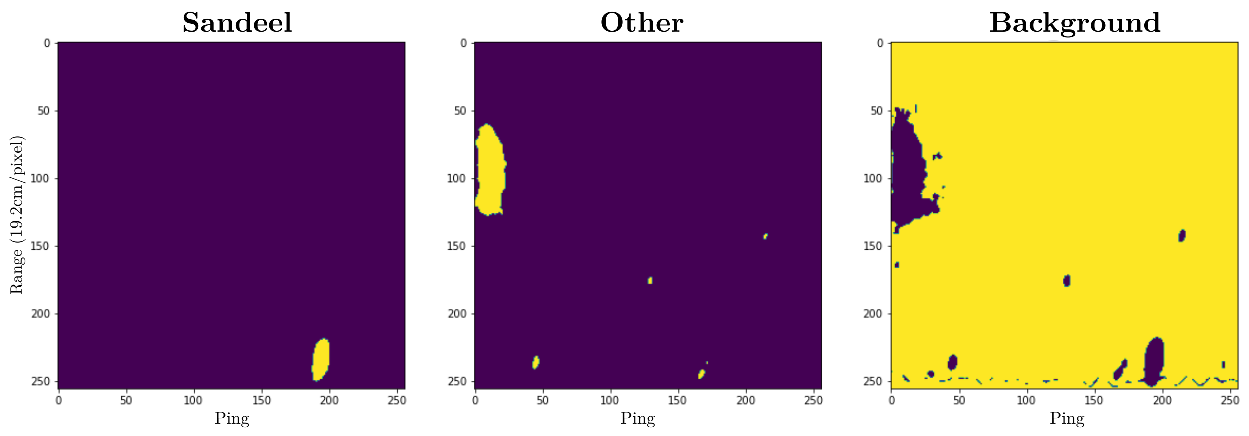


Figure 4.3: *An example pseudo label displaying the hard mask of all three classes present in a 256×256 crop: Sandeel, Other, and Background.*

4.4 Data Preparation

This section explains the process of preparing the dataset, which was built to enable a sampling scheme during training equal to the one developed by Brautaset et al. [3]. The data preparation consists of creating samples with corresponding pseudo labels from the input s_v data and storing the samples in a folder structure dependent on the sample's features.

The entire process is illustrated in figure 4.4 and starts with utilizing the CRIMAC preprocessor module as described in 4.2. This takes in the entire *.raw* dataset and outputs the s_v data in the *.zarr* format. The s_v dataset was then sent to both the pretrained U-Net and bottom detection modules from the CRIMAC pipeline. The pretrained U-Net outputs pseudo labels, and the bottom detection outputs a mask of the seafloor as arrays equal in spatial size to the s_v data array.

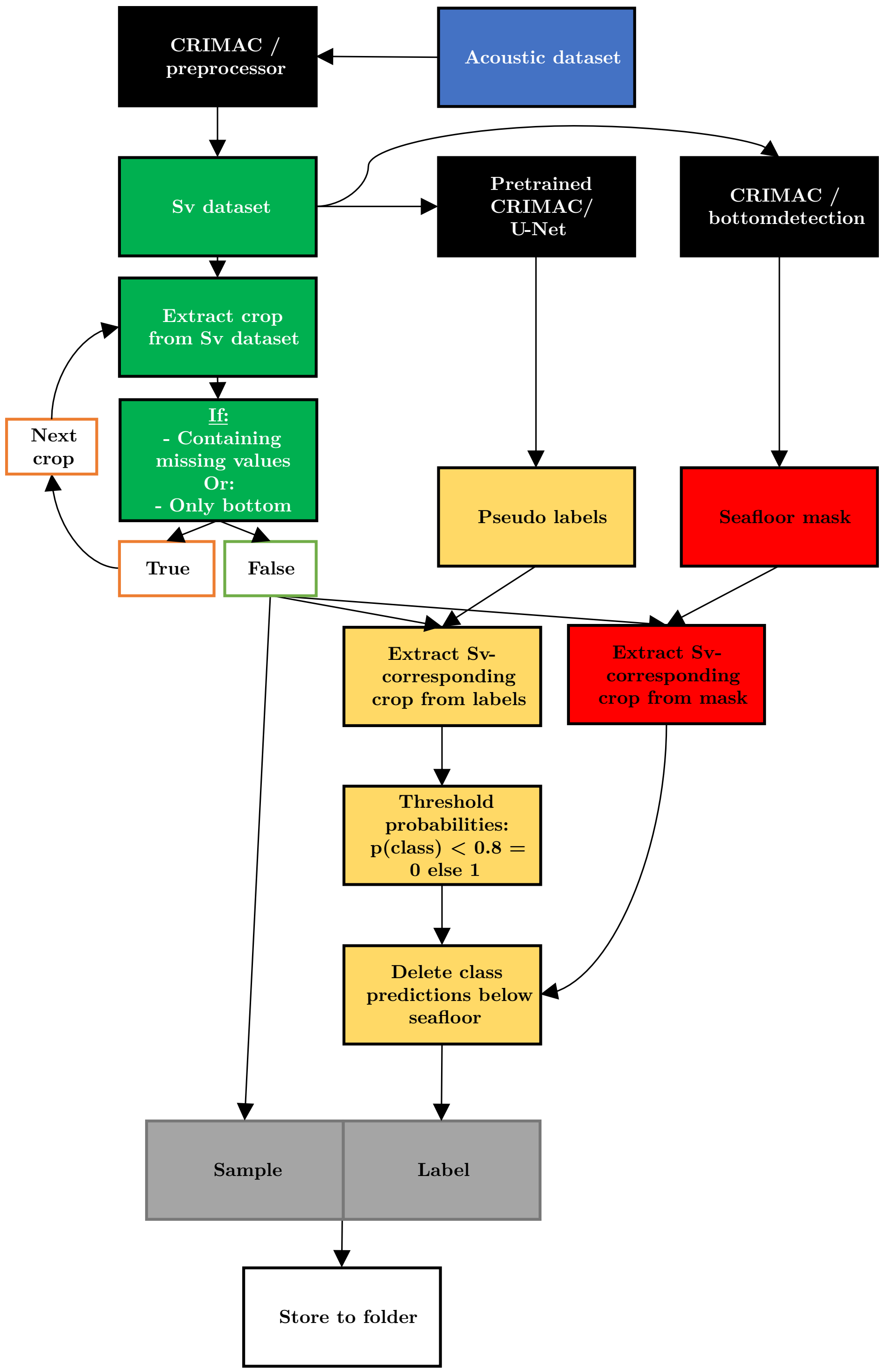

CRIMAC / preprocessor
Acoustic dataset
Sv dataset
Pretrained CRIMAC/ U-Net
CRIMAC / bottomdetection
Extract crop from Sv dataset
Next crop
If:
- Containing missing values
Or:
- Only bottom
True
False
Pseudo labels
Seafloor mask
Extract Sv-corresponding crop from labels
Extract Sv-corresponding crop from mask
Threshold probabilities: p(class) < 0.8 = 0 else 1
Delete class predictions below seafloor
Sample
Label
Store to folder

Figure 4.4: *An overview of the data preparation process. CRIMAC pipeline modules (black), data.raw (blue), s_v data.zarr (green), pseudo labels.zarr (yellow), bottom.zarr (red), the finished file containing two tensors (gray), storing the file with Python pickle (white).*

The generated s_v dataset was then split into non-overlapping crops of size 256x256, ignoring crops of mismatching size at the edges of the array. Each crop was checked for any missing values or the crop being located entirely below the seafloor and then ignored if any of them were true. As visualized in figure 4.5 there were instances of discontinuity in both the time series of pings and range. This caused portions of the s_v dataset to be filled with missing values, hence motivating the previous check as sampling crops at complete random could cause errors.

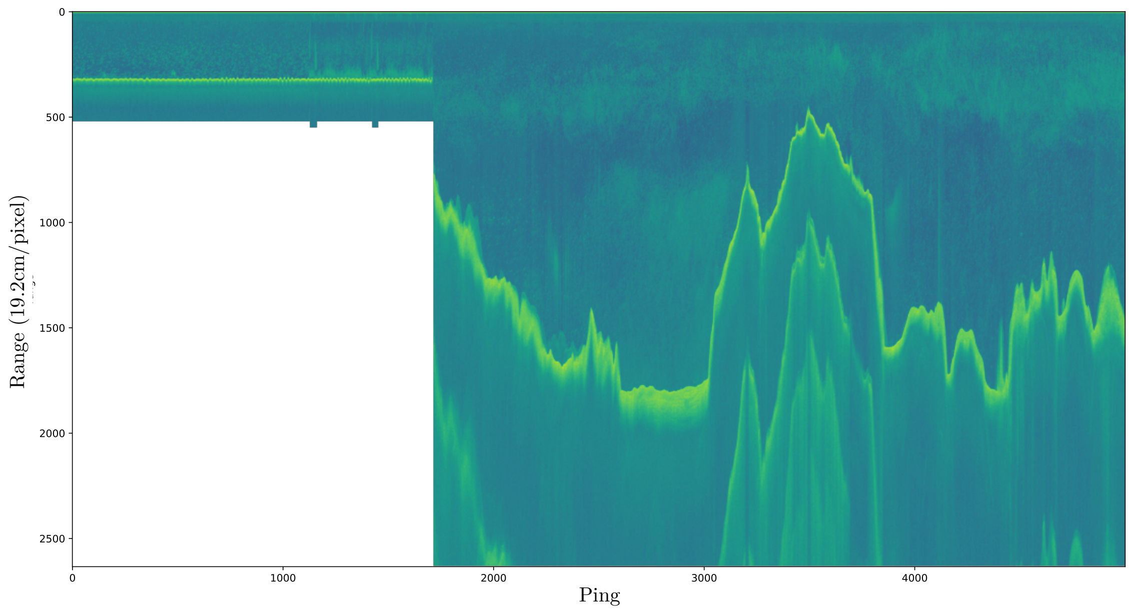


Figure 4.5: *Example crop from the 18kHz s_v data (decibel scale). It illustrates two clear sections of data where the depth (range) is changed, and the gap is filled with missing values (white). The bottom can be observed by the strong line of s_v values, and there are multiple bottom echoes in certain parts of the crop.*

Using the vertical and horizontal coordinates for all the s_v data crops, a corresponding crop from both the seafloor mask and the pseudo labels was extracted. The two new crops were then used together to remove predictions that appeared under the bottom, thus cleaning and preprocessing the pseudo labels. The steps explained in this paragraph are also visualized in figure 4.6.

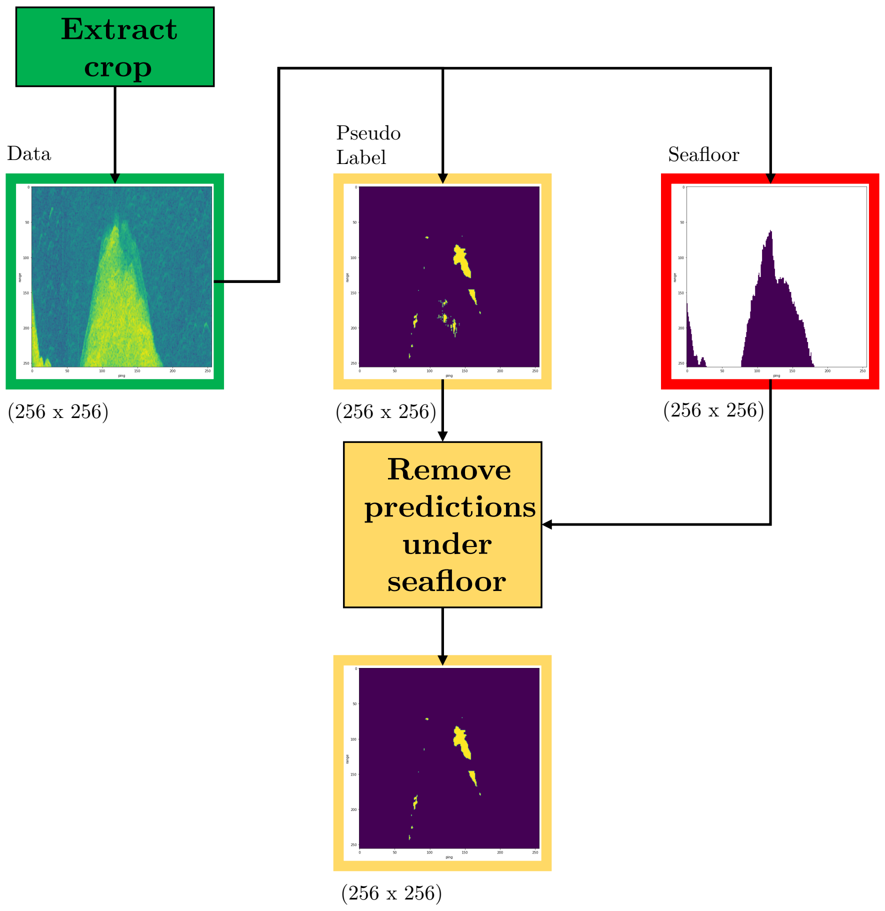


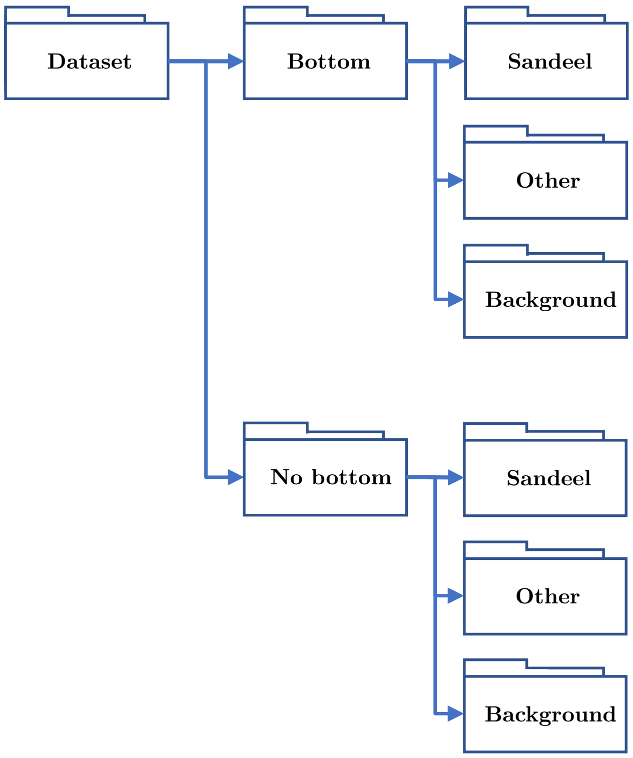
Figure 4.6: *Example of how the data, pseudo labels, and bottom crops look during data generation. For the pseudo label, purple is values of 0 and yellow values of 1. In the pseudo labels, we can see that some predictions under the seafloor are removed. Size is shown in the lower-left corner to clarify that it is a crop of the same size and from the same location but from different arrays.*

.

After the crop containing the s_v data and cleaned pseudo labels were generated, they were both converted to tensors and stored as a single file using Python pickle. The storage folder of the file depend on a set of criteria, which are illustrated in figure 4.7. The criteria were checked from top to bottom, and the first triggered set the destination folder, creating a folder structure visualized in figure 4.7. This also blocked the same crop from appearing in several folders, potentially causing data leakage between datasets.

Criteria:
1. **100+ pixels == sandeel class, no bottom**
2. **100+ pixels == sandeel class + bottom**
3. **100+ pixels == other class, no bottom**
4. **100+ pixels == other class + bottom**
5. **Contains background class and bottom**
6. **Contains background, no bottom**

(a) *The list of criteria.*



(b) *The folder structure.*

Figure 4.7: *An overview of the criteria and folder structure. The structure consists of two main branches, one with the bottom present and one without the bottom present.*

The folder structure and criteria were based on the data classes made by Brautaset et al. [3]. One significant difference was that they used the actual operator annotations to detect instances of classes in a crop, while we only have the pseudo labels. Since these pseudo labels could contain misclassifications made by the pretrained model, a class abundance measurement was performed. When evaluating the crops against the criteria, if 100 or more pixels in a crop contained either *sandeel* or *other*, the class was set to be present in the crop. This removed crops with low class abundance and where the pretrained model had potentially misclassified single or few pixels. If there were not enough pixels of either class, the crop was set to contain *background* only. The crop from the s_v data was checked against its corresponding crop from the bottom detector output, splitting the data into folders with or without the bottom present.

The years 2018 and 2019 were processed in the way mentioned above individually, and the final data distribution can be seen in respectively table 4.1 and 4.2.

Table 4.1: *Data distribution for the 2018 dataset.*

2018 dataset		Occurrences	% of dataset
	Sandeel	1410	7.3%
No bottom	Other	184	0.95%
	Background	6519	33.75%
	Sandeel	1297	6.71%
Bottom	Other	840	4.35%
	Background	9067	46.94%
Total:		19317	100%

Table 4.2: *Data distribution for the 2019 dataset.*

2019 dataset		Occurrences	% of dataset
	Sandeel	1308	4.38%
No bottom	Other	1074	3.60%
	Background	10291	34.45%
	Sandeel	1652	5.53%
Bottom	Other	2815	9.42%
	Background	12733	42.62%
Total:		29873	100%

From the tables above, we observe that, as in Brautaset et al. [3], most of the data will contain no fish, here represented as *background*.

4.5 Experiment

This section describes how the experiments were performed and how performance was measured, first by describing the settings, then the experiment itself in detail.

4.5.1 Experiment Settings

The experiment uses the same U-Net architecture as described in section 3.1 with the alteration developed by Brautaset et al. [3]. The change to the architecture in this work was to adjust the number of frequency channels in the input layer. This ranged from one to six, depending on the number of frequencies in a subset. All machine learning was implemented using the Python library PyTorch [44].

Data loaded to the model started with first selecting a folder from the folder structure, with a set probability for each. Then a sample and a label was extracted at random from this folder, with all samples in each folder being allocated to either training, validation, or test datasets. This imitates the sampling strategy from Brautaset et al. [3], and the probabilities originate from this work.

Table 4.3: *The sample classes correspond to the folder structure described in figure 4.7. Each is given a probability of being the target folder for sample extraction.*

Sample class	**Probability**	**Details**
Sandeel	5/26	Random crop containing the sandeel class
Other	5/26	Random crop containing the other class
Background	1/26	Random crop containing no fish
Sandeel + bottom	5/26	Random crop containing the sandeel class and bottom
Other + bottom	5/26	Random crop containing the other class and bottom
Background + bottom	5/26	Random crop containing no fish and bottom

The data-augmentations performed were flipping along the vertical axis, and a multiplication applied to values in 5% of pixels in an input s_v crop to act as noise. An echogram's orientation of the surface and bottom can never individually be changed during training, so only vertical flipping is justified in acoustic data. The intention behind the multiplication was to simulate noise and consisted of multiplying the values with a random number in either the range [0,1] or [1, 10], with a 50% chance of each. As samples from the training data were provided to the model, each data augmentation method had a 50% chance of occurring. Then the s_v crop was transformed to the decibel scale by applying $10 \log(pixel)$ to all pixels for all frequencies in the s_v crop, with cutoff values at minimum -75dB and maximum 0dB. All augmentations and transformations were based on Brautaset et al. [3].

Table 4.4: *Description of each data augmentation performed.*

Data augmentation	**Details**
Add noise to 5% of pixels.	50% of occurring upon loading sample
Flip along the vertical axis	50% of occurring upon loading sample

The hyperparameters used during training are summarized in table 4.5 and equal those used by Brautaset et al.[3].

Table 4.5: *Settings for all hyperparameters used during the training.*

Hyperparameters	**Value/ Category**	**Details**
Loss function:	Weighted Cross-entropy	Background = 1, Other = 25, Sandeel = 30
Optimizer:	Stochastic gradient descent	
Learning rate:	0.01	Halved every 1000th batch
Momentum:	0.95	
Mini-Batch size:	16	
Crop size:	256×256	Include all available frequency channels

To evaluate the performance of the models, we utilized the F1 score mentioned in section 2.5.1. We also included the precision and recall, as these provide greater insight into the model's performance.

4.5.2 Exhaustive Frequency Search

Description

This experiment would see all combinations (*subsets*) of all frequencies being evaluated. Each subset of frequencies would instantiate a new U-Net model based on the architecture described in section 4.5.1, with input channels equal to the number of frequencies in the subset. The experiment is similar to and inspired by KD methods and train student models (*the model instantiated by a subset*) on the data labeled by the teacher (*baseline model*). The performance of each subset will be estimated by how well it reconstructs the test data labeled by the teacher. The experiment was run *ten* times to get enough results to estimate the mean, reduce variance, and display statistical data for each subset of frequencies. With six frequencies in total, the number of possible frequency subsets for each experiment is 63[1]. During training, only the frequency channels present in the subset would be extracted from the training data and provided to the model. The ten experiments had different random state seeds set to enable different data splits and variable model initialization. The explicit random state also accommodates reproducibility.

The dataset for the year 2019 in table 4.2 contained the most data and was chosen for training. 30% of the training dataset was assigned as the validation dataset. Meanwhile, the 2018 dataset was selected as the test dataset. The total amount of data given to a model during training was 5,000 batches. With a mini-batch size of 16, this corresponds to 80,000 samples, which matches the amount of data given to the model by Brautaset et al. [3].

[1]Calculated as combinations without repetitions and without order, using formula $\frac{n!}{k!(n-k)!}$, where n is elements to chose from and k is how many to choose, both values as integers. Summarize for k in range 1..6.

Monitoring Training

Logging of metrics occured at different stages during the training process;

- Every mini-batch: Calculate the exponential moving average of the training loss.
- Every mini-batch: Log the exponential moving average of the training and validation loss.
- Every 250 mini-batch: Run model on 100 mini-batches from the validation data, plot one sample output and its label, and calculate exponential moving average loss on each validation minibatch. Furthermore, calculate F1-score on both training and validation data for where the sandeel class is positive, and the other classes are the negative.

The training process was split into 50 epochs to accommodate this logging scheme, with 100 mini-batches in each. This epoch is only related to the logging scheme, not the data itself. Hence, each epoch is not the entire dataset but a part of a sequence of data. This is mentioned to not confuse a reader observing the results. The metrics loss and F1-score indicate convergence, over-/under-fitting problems, and performance. A sanity check was also performed by observing and judging a prediction of the sandeel class against its label.

Generalization Performance per Subset

After the training of a frequency subset, the subset's model was evaluated on 500 batches from the test dataset but without any noise added through augmentation. From these predictions, the F1-score, precision, and recall were calculated for where sandeel was the positive class and the two other classes were negative, giving an estimate of its generalization performance. Then the next frequency subset started the entire process afresh with training.

With the results from the ten separate experiments, each on all 63 possible combinations, each frequency subset's mean test performance (*generalization performance*) values were estimated. This was then used to show these properties:

- A summary of the performance achieved by each subset size.
- Finding the best performing frequency combinations per subsets size. One max search based on each metric; F1-score, precision, and recall.

- To observe the uniqueness of each subset, a plot to illustrate the statistical properties of each will be presented. This will be shown as error bars, the top of the bars is the max F1-score achieved, and the bottoms is the minimum.
- The performance trend of each frequency. This meant, for all frequencies, filtering out those subsets it was a part of and sorting these in increasing order. Producing a line plot illustrating what performance scores each particular frequency contributes to and its overall trend.

4.6 Hardware

Two remote servers named *Janus* and *Birget* were provided by the University of Bergen. Janus was used for data preparation and Birget for conducting the experiments. Hardware spesifications:

- **Birget**
  - CPU: AMD EPYC 7742 64-Core Processor
  - GPU: 8 x A100-SXM-80GB (*One available for this thesis*)
- **Janus**
  - CPU: Intel(R) Core(TM) i9-7900X CPU @ 3.30GHz
  - GPU: 2 x GeForce RTX 2080 Ti (*One partially available for this thesis*)

Chapter 5

Results

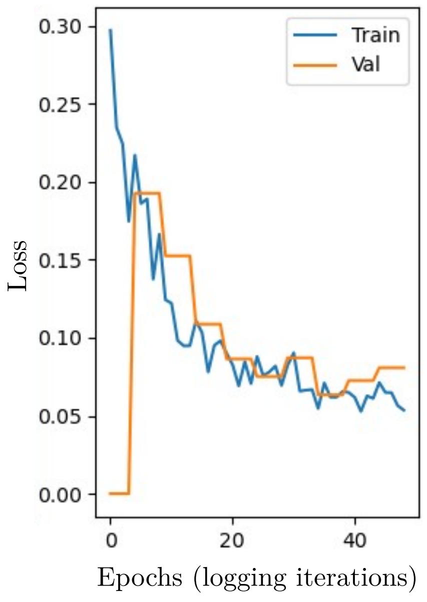
This chapter will first describe the monitoring and evaluation of the training process. Then a summary of the exhaustive search, followed by the subsets responsible for the highest mean performance per subset size. All individual subsets' performances are visualized to find the unique subsets. Finally, the performance trend for each frequency is shown.

5.1 Experiment: Exhaustive Search

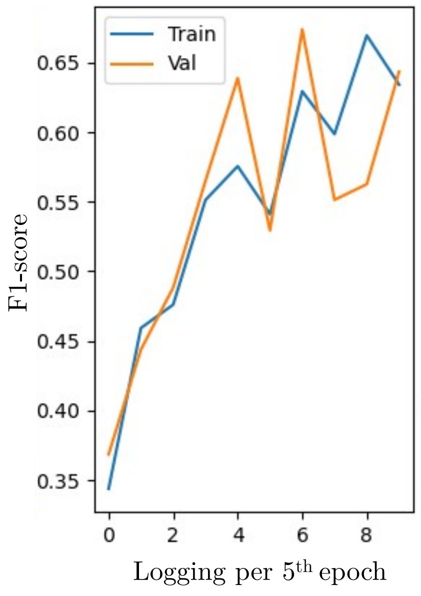
5.1.1 Training

Figure 5.1 illustrates the metrics measured during the training of a model using a subset consisting of all six frequencies. It shows that both the validation and training loss follow the same trend, showing convergence and no issues regarding over-/under-fitting. This is roughly mirrored by the F1-score, as the validation and training F1-scores follows approximately the same values. Finally, the visual inspection suggests satisfactory performance of the classifier on the *sandeel* class, as illustrated in figure 5.2.

Some combinations with one or two frequencies likely needed more training to converge by viewing the complete logs from training. More examples can be found in appendixes B.3.



(a) *The logged loss.*



(b) *The logged F1-score.*

Figure 5.1: *The F1-score and loss for both training and validation. The validation loss was calculated less often and instanciated as 0, resulting in more significant jumps in value and its low initial value. The training and validation F1-score on the final validation data were respectively 0.63 and 0.64.*

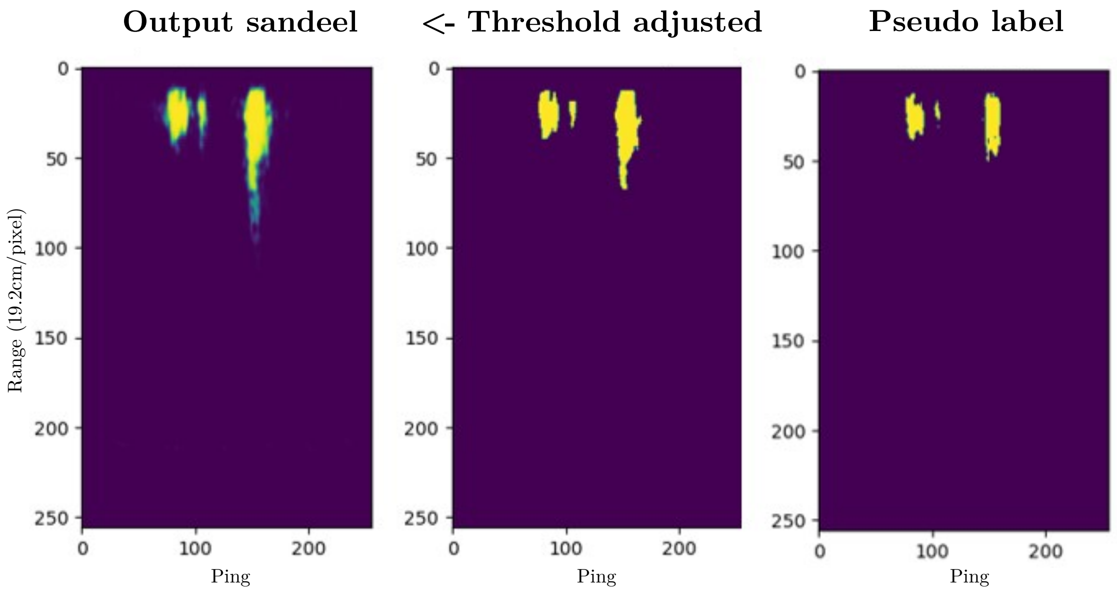


Figure 5.2: *From left to right; Network output for the sandeel class, same output threshold adjusted with 0.8, and finally, the label for the current sample.*

5.1.2 Results: Exhaustive Search

This section starts with a summary of the subsets' mean F1-score from the ten experiments. They were summarized per subset size and show minimum F1-score, maximum F1-score, and median F1-score in table 5.1. The minimum and median scores increase with the subset size, but the max performance stagnates after a subset size of three is reached. The mean F1-score of all individual subsets can be viewed in appendix B.1.

Table 5.1: *This table summarizes the mean performances for each size of subsets over the ten tests.*

Subset size:	**1**	**2**	**3**	**4**	**5**	**6**
F1-score (min):	0.25	0.28	0.34	0.42	0.50	0.66
F1-score (median)	0.28	0.37	0.45	0.54	0.62	0.66
F1-score (max):	0.34	0.46	0.65	0.67	0.67	0.66

Figure 5.3 illustrate the exact frequencies contributing to the max mean F1-score for each subset size. A clear jump in performance can be seen when the subset size increase from one to two, but most significantly when the subsets *18kHz*, *38kHz*, and *200kHz* are used.

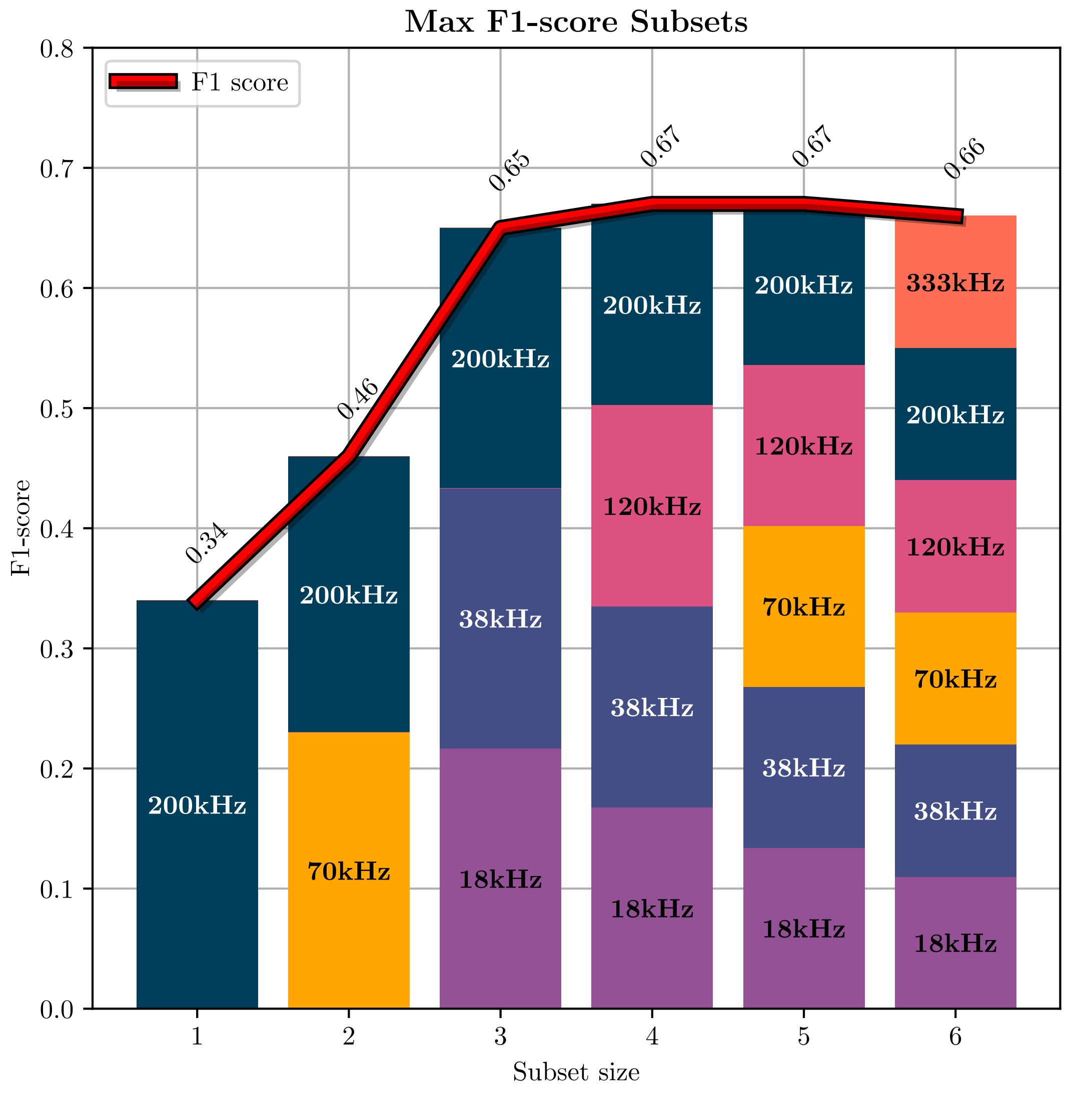


Figure 5.3: *The red line shows the max F-score achieved for each subset size. The blocks indicate only which frequency was present in the subset achieving this F1-score, and does not indicate in what magnitude the frequencies contributed to the score.*

The exact frequencies contributing to the max performing subsets are visualized for precision in figure 5.4. It follows the same trend as seen in figure 5.3 for F1-score but with lower scores.

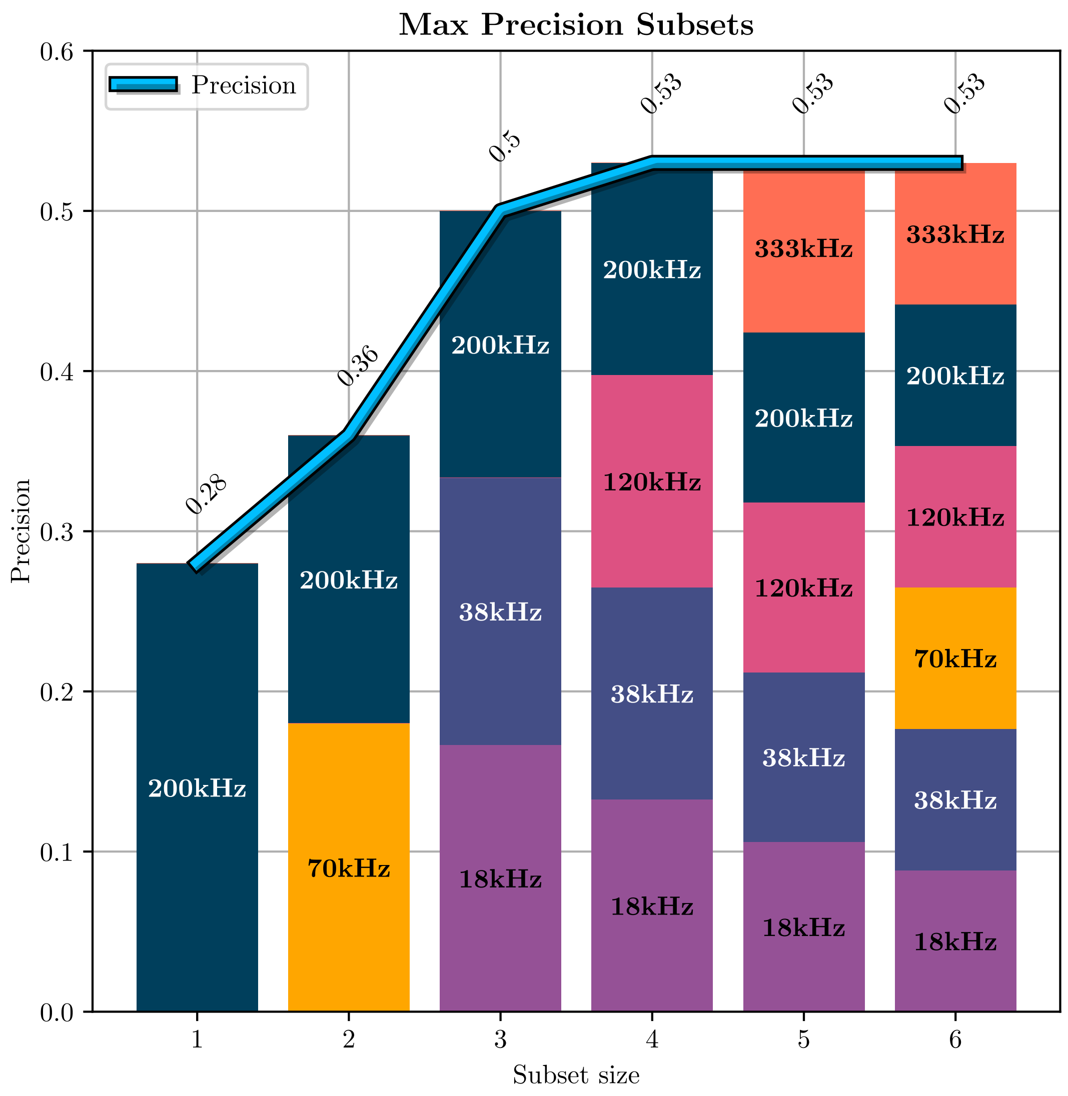


Figure 5.4: *The blue line shows the max precision achieved for each subset size. The blocks indicate only which frequency was present in the subset achieving this precision, and does not indicate in what magnitude the frequencies contributed to the score.*

The exact frequencies contributing to the max performing subsets are visualized for recall in figure 5.5. It reaches high values of recall at a subsets size of two. Recall is also always higher than precision for all subsets sizes, showing that recall plays the most significant role in the F1-score.

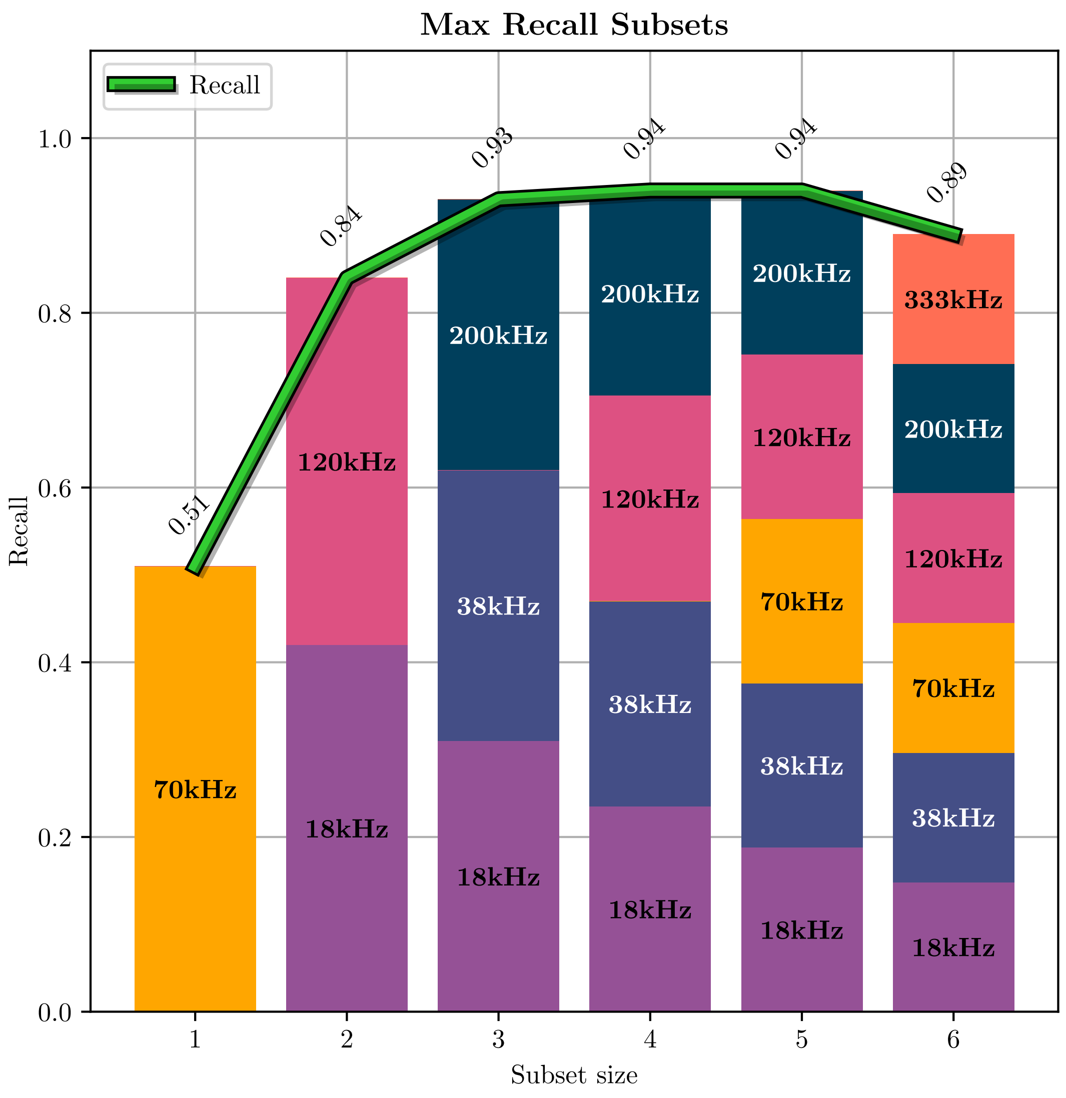


Figure 5.5: *The green line shows the max recall achieved for each subset size. The blocks indicate only which frequency was present in the subset achieving this recall, and does not indicate in what magnitude the frequencies contributed to the score.*

5.1.3 Individual Subsets

Figure 5.6 illustrates a complete plot of all error bars, with sections for each subset size. A red circle encompassing the error bar related to the subset containing only *18kHz*, *38kHz* and *200kHz*. This combination outclasses all other subsets in the same and previous subset sizes and competes with all the results later achieved by larger subsets. In all other sections, no subset stands out regarding F1-score.

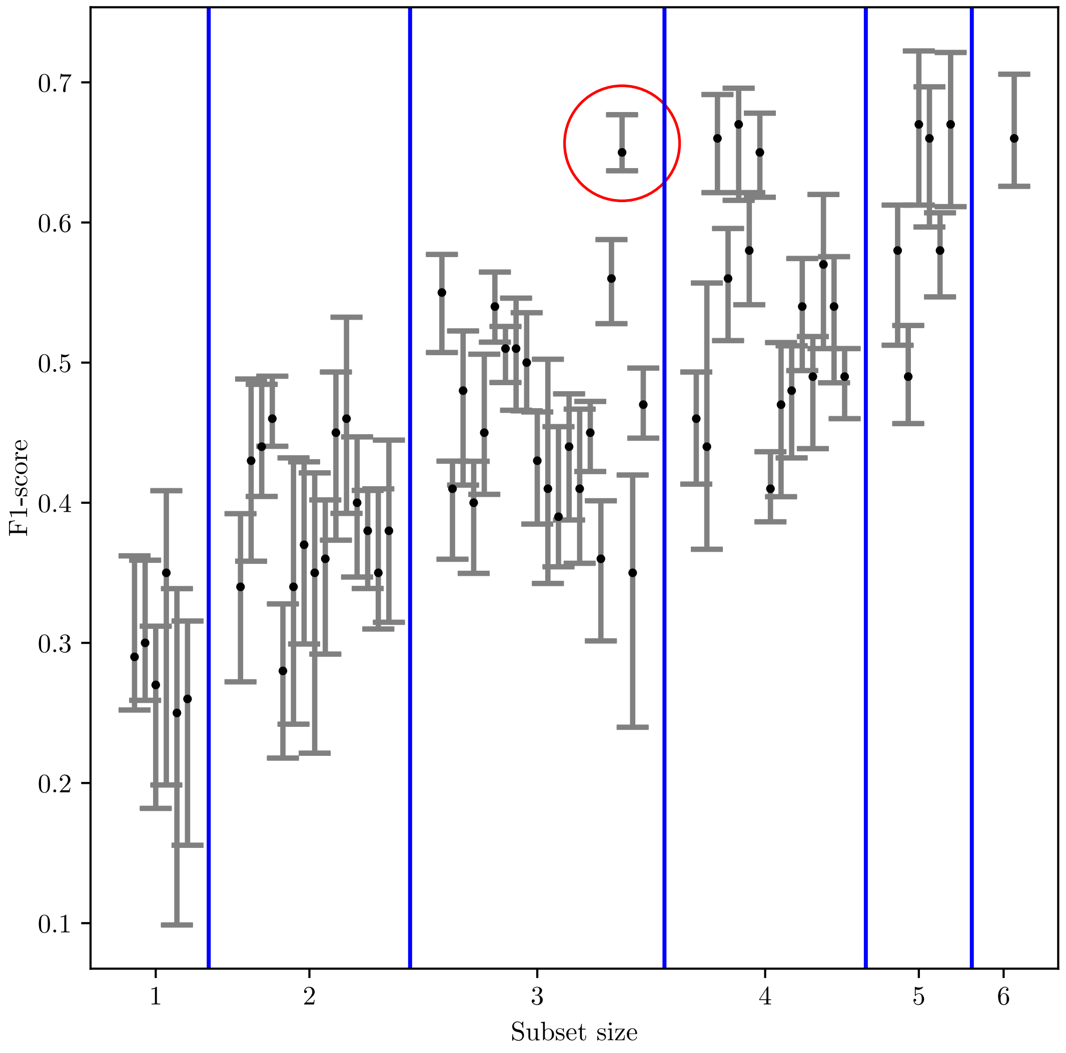


Figure 5.6: *Each error bar represents a frequency subset. The error bar's highest y-value is the max value achieved for this subset during all ten tests, and the bottom is the lowest. Blue vertical lines group the error bars by the size of the subset and the black dots are the mean performance of the subset. A red ring encompasses the subsets 18kHz, 38kHz, and 200kHz.*

To further analyze the performance of the unique subset *18kHz*, *38kHz*, and *200kHz*, two new subsets based on the results were created. The first with subsets that, at a minimum, contained the aforementioned frequencies (*eight in total*). These subsets were then removed from the initial results, and then the eight best performing from this set were extracted. Both of these subsets are visualized in figure 5.7 and show that all the highest performing subsets include the frequencies *18kHz*, *38kHz*, and *200kHz*.

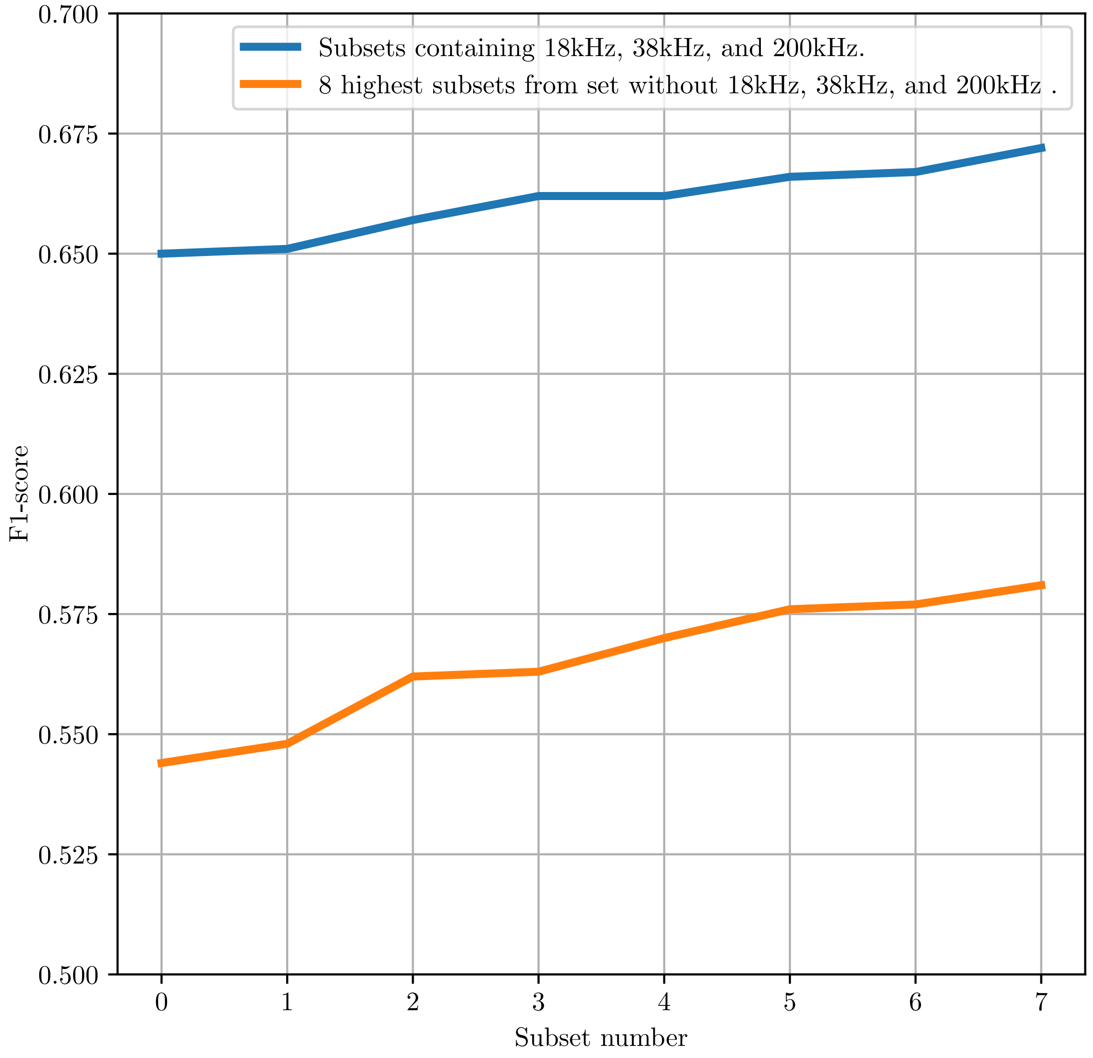


Figure 5.7: *In this figure, the mean F1-score of all subsets containing 18kHz, 38kHz, and 200kHz is plotted as a blue line (Eight in total). The orange line is the eight highest performing subsets from a set where the subset 18kHz, 38kHz, and 200kHz is not included in any subset.*

5.1.4 Performance Trend per Frequency

In this section, the performance trend for each frequency is visualized. For each frequency, all subsets it was part of were extracted and then sorted in increasing order of mean F1-score. Each frequency trend is visualized in figure 5.8. From right to left, the plot shows that *18kHz*, *38kHz*, and *200kHz* all are part of many combinations with high F1-scores. The remaining frequencies quickly fall in performance. Furthermore, of all frequencies, the line associated with 18kHz have the highest overall trend regarding F1-score. Which is the opposite for 120kHz which have the worst performance trend, followed closely by 70kHz.

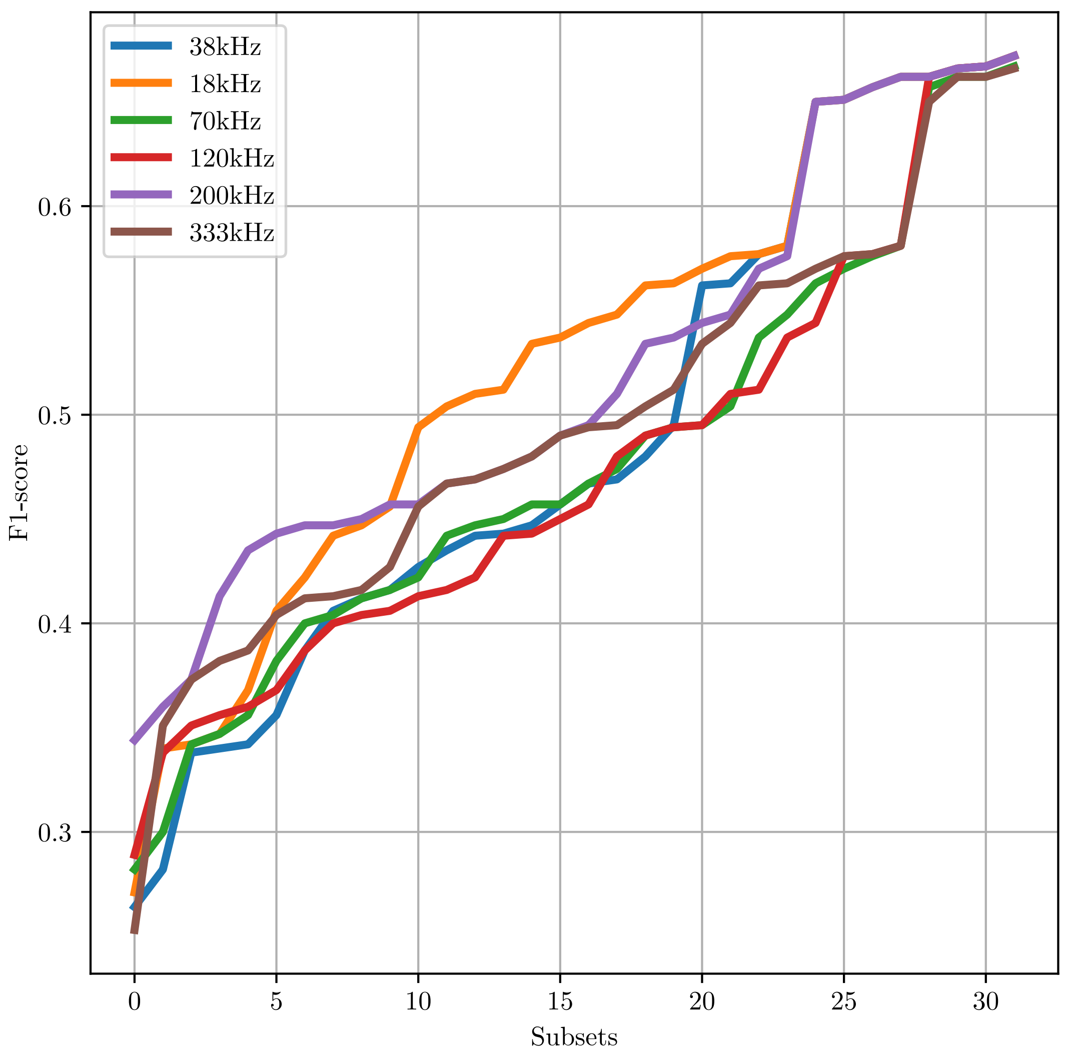


Figure 5.8: *This figure illustrates the performance trend of each frequency. For each frequency (color above), all subsets that contain that frequency were extracted from the results and sorted in increasing order of mean F1-score, then visualized. A total of 32 subsets for each frequency. When lines merge they achieved the same F1-score, but may not represent the same subset.*

Chapter 6

Discussion

In this thesis, an approach to find the subset of frequencies most applicable to classify sandeel in acoustic data has been implemented to advise how to equip smaller unmanned vessels. The results in figure 5.3 illustrate the best performing combination of frequencies per subset size and would act as advice for vessels that are to be equipped with a subset of echo sounders using the frequencies tested. After the subset size of three, the performance does not increase significantly for larger subsets in F1-score. The recall is the most significant part of the F1-score, which implies that finding instances of possible sandeel in the data requires fewer frequencies than accurately classifying it, which is to be expected as the latter is likely a more complex task requiring additional information.

The unique performance of the subset containing *18kHz*, *38kHz*, and *200kHz*, depicted in figures 5.3 and 5.6, supports the current methods applied by the IMR described in section 1.4, where the same frequencies are important for the classifications (*18kHz* and *38kHz*) and delineation (*200kHz*) of sandeel schools. However, our results cannot tell *how* and in *what magnitude* each of the frequencies contributes to the classification process for this subset, a clear possibility for future work. The *200kHz* channel is present in all subsets in figure 5.3, and for the subsets size of two, the *70kHz* is chosen over both *18kHz* and *38kHz*[1]. This suggests a significant positive synergy between the two latter frequencies and 200kHz as they outperform all other subsets at size three. Furthermore, figure 5.7 shows that the unique subset of three (*18kHz, 38kHz, and 200kHz*) is part of all the highest performing subsets, and

[1]Although with a small margin to the other subsets at subset size two as described in appendix B.2.

a sharp drop in performance can be seen if they are not all present in a subset. This leads to the proposition that vessels equipped with three or more echo sounders should include *18kHz*, *38kHz*, and *200kHz* as a minimum.

The generalizability of this work was likely reduced by the amount of data used during the exhaustive search and was restricted because of little computational power. In the works of Brautaset et al., they utilized a total of 12 years, while this work only used data from two years (*2018 and 2019*). On entire echograms, their highest F1-score was 0.78 in 2018 (*our test dataset*), while the best subsets in this thesis resulted in a mean performance of 0.67. The performance achieved by the models in this work showed that some could generalize from 2019 to 2018 on a much smaller training dataset than used in Brautaset et al. [3]. The results from Brautaset et al. may indicate that the quality of the data itself increased in the more recent years. As the data quality is crucial for the model, this may indicate that acoustic classifiers can be successfully trained using far fewer quantities of high-quality data. This assumes that the data contained in the 2019 dataset is high-quality, which is hard for us to evaluate. Meanwhile, more data will likely make the classifier more robust to noise. Future work should retrain and test our model on the same years and amounts of data described in Brautaset et al. to establish a more comprehensive comparison.

The original annotations belonging to the acoustic data were unavailable during this thesis; thus the solution became KD applied through pseudo labels. This was considered befitting, as the objective of this task was not to train a better model but assess which frequencies explain the performance achieved by the state-of-the-art baseline model created by Brautaset et al. [3]. The teacher-student approach was justified by the solid performance shown by the baseline model, especially on the year 2018. Thus, the pseudo label would likely sufficiently capture the baseline model's use of the frequencies, and new models trained using the pseudo labels would provide information regarding the performance of the subsets relative to the other subsets. Meanwhile, this would mean that the models trained in this thesis would likely never surpass, just approach the baseline models performance. As models produced in this work have shown performance close to Brautaset et al., the results were judged to be adequate to formulate advice for the maximally informative subset of frequencies. Future work should investigate alternative KD approaches, such as changing the threshold of 0.8 applied to create the hard mask for the pseudo labels or use soft labels instead. An ensemble of teachers models could also been used during training, for example using the baseline model, the traditional benchmark method applied in Brautaset et al. created by Korneliussen et al.

[27], and the two newer methods proposed by Marques et al. [33] and Choi et al. [4] described in section 3.2. Additionally, changes should be made to allocate all pixels to the background class, which was not assigned to the sandeel or other classes after the threshold. The latter would likely not significantly affect the results, as most pixels were allocated to a class.

There is likely some bias toward the frequencies used in the basis work. The features the additional two frequencies (*70kHz and 333kHz*) used in this work may have contained were not included when training the basis model: thus, they played no part in forming the pseudo labels. However, by viewing figure 5.8, *120kHz* can be interpreted as a frequency present in a high number of low-performing subsets. The performance trend was similar to *70kHz* but worse than *333kHz*. This suggests that the information found in *120kHz* may also be found in *70kHz* and *333kHz*. The *18kHz* had the highest trend in the same figure, but it was not present in a subset before a subset size of three in figure 5.3, measuring max F1-score. This suggests that *18kHz* has a positive synergy with many other frequencies but performs poorly in small subsets. Further work could look into removing the *120kHz* frequency, rerun the experiments of Brautaset et al. [3], and observe the change in performance. Possibly discovering that their performance was reliant on the same unique subset (*18kHz, 38kHz, and 200kHz*) found in this thesis. Additionally, the importance of the 18kHz frequency should be further investigated as this is a heavy transducer, which means that the unmanned vessels need to have the structural capacity to carry it.

Additionally, information about the range from the transducer in the crops evaluated by the model would likely increase performance. As acoustic backscattering from the same object can change with range, providing the model with additional context regarding the crop's range would likely provide a better foundation for handling this complexity. Range should be evaluated as a feature to include in future work when applying CNNs to acoustic data crops.

Chapter 7

Conclusion and Future Work

In this thesis, acoustic classifiers were successfully trained on pseudo labels with varying subsets of frequencies. The performance of each subset was measured to advise on how to equip lightweight unmanned vessels with echo sounders to monitor sandeel. Based on our results, one can conclude that the information gained when selecting the maximally informative frequency subsets increase from a size of one up to two, and drastically to three. However, from a size of three and afterward the subsets depend on the frequencies *18kHz*, *38kHz*, and *200kHz* to maximize information. Considering the size of transducers operating at these frequencies, the unmanned kayak introduced in section 1.5 is likely too small to carry this subset and larger unmanned vessels should be considered when classifying sandeel.

In future research, we propose that the experiment created in this thesis should be tested and trained on additional data and with new features. Only the data from 2018 was used as test data during this work, and our results should be verified across years. Especially those years used in the basis work of Brautaset et al., focusing on verifying the performance of subset: *18kHz*, *38kHz*, and *200kHz*. In addition, the importance of the 18kHz should be further investigated as this is a large transducer. The method implemented to create the pseudo labels can be further optimized and new avenues within KD should be tested. Additionally, we propose the implementation of a ”range” feature in our model to provide the CNN with additional context for each crop.

List of Acronyms and Abbreviations

ANN	artificial neural networks.
CNN	convolutional neural network.
CRIMAC	Center for Research Based Innovation in Marine Acoustic Abundance Estimation and Backscatter Classification.
ICES	International Council for the Exploration of the Sea.
IMR	Norwegian Institute of Marine Research.
KD	Knowledge distillation.
LSSS	Large Scale Survey System.
MLP	multi-layer perceptron.
$\mathbf{s_v}$	volume backscattering coefficient.
SGD	stochastic gradient descent.
TS	target strength.

Bibliography

[1] Sajjad Abbasi, Mohsen Hajabdollahi, Nader Karimi, and Shadrokh Samavi. Modeling teacher-student techniques in deep neural networks for knowledge distillation. In *2020 International Conference on Machine Vision and Image Processing (MVIP)*, pages 1–6. IEEE, 2020.

[2] Abdolmaged Alkhulaifi, Fahad Alsahli, and Irfan Ahmad. Knowledge distillation in deep learning and its applications. *PeerJ Computer Science*, 7, 2021.

[3] Olav Brautaset, Anders Ueland Waldeland, Espen Johnsen, Ketil Malde, Line Eikvil, Arnt-Børre Salberg, and Nils Olav Handegard. Acoustic classification in multifrequency echosounder data using deep convolutional neural networks. *ICES Journal of Marine Science*, 77(4):1391–1400, 2020.

[4] Changkyu Choi, Michael Kampffmeyer, Nils Olav Handegard, Arnt-Børre Salberg, Olav Brautaset, Line Eikvil, and Robert Jenssen. Semi-supervised target classification in multi-frequency echosounder data. *ICES Journal of Marine Science*, 78(7):2615–2627, 2021.

[5] Changkyu Choi, Michael Kampffmeyer, Nils Olav Handegard, Arnt-Børre Salberg, Olav Brautaset, Line Eikvil, and Robert Jenssen. Semi-supervised target classification in multi-frequency echosounder data. *ICES Journal of Marine Science*, 78(7):2615–2627, 08 2021. ISSN 1054-3139. doi: 10.1093/icesjms/fsab140.
URL: `https://doi.org/10.1093/icesjms/fsab140`.

[6] Wikimedia Commons. File:amphiprion ocellaris (clown anemonefish) by nick hobgood.jpg — wikimedia commons, the free media repository, 2020.
URL: `https://commons.wikimedia.org/w/index.php?title=File:Amphiprion_ocellaris_(Clown_anemonefish)_by_Nick_Hobgood.jpg&oldid=446928047`. [Online; accessed 21-April-2022].

[7] Wikimedia Commons. File:ammodytes hexapterus.jpg — wikimedia commons, the free media repository, 2020.
URL: `https://commons.wikimedia.org/w/index.php?title=File:Ammodytes_hexapterus.jpg&oldid=477294458`. [Online; accessed 21-April-2022].

[8] Marshall Copeland, Julian Soh, Anthony Puca, Mike Manning, and David Gollob. *Microsoft Azure: Planning, Deploying, and Managing Your Data Center in the Cloud.* Apress, USA, 1st edition, 2015. ISBN 1484210441.

[9] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2016.

[10] Wissal Farsal, Samir Anter, and Mohammed Ramdani. Deep learning: An overview. In *Proceedings of the 12th International Conference on Intelligent Systems: Theories and Applications*, pages 1–6, 2018.

[11] Aram Galstyan and Paul R Cohen. Empirical comparison of "hard" and "soft" label propagation for relational classification. In *International Conference on Inductive Logic Programming*, pages 98–111. Springer, 2007.

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016. `http://www.deeplearningbook.org`.

[13] Arne Hassel, Tor Knutsen, John Dalen, Kristian Skaar, Svein Løkkeborg, Ole Arve Misund, Øivind Østensen, Merete Fonn, and Eli Kyrkjebø Haugland. Influence of seismic shooting on the lesser sandeel (ammodytes marinus). *ICES Journal of Marine Science*, 61(7):1165–1173, 2004.

[14] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[15] Yaoshiang Ho and Samuel Wookey. The real-world-weight cross-entropy loss function: Modeling the costs of mislabeling. *IEEE Access*, 8:4806–4813, 2019.

[16] S. Hoyer and J. Hamman. xarray: N-D labeled arrays and datasets in Python. *Journal of Open Research Software*, 5(1), 2017. doi: 10.5334/jors.148.
URL: `https://doi.org/10.5334/jors.148`.

[17] ICES. Guide to ICES advisory framework and principles. 12 2020. doi: 10.17895/ ices.advice.7648.
URL: `https://bit.ly/3rPqs38`.

[18] ICES. Greater North Sea Ecoregion – Ecosystem overview. 12 2021. doi: 10.17895/ ices.advice.9434.
URL: `https://bit.ly/3kbZJJK`.

[19] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR.
URL: `https://proceedings.mlr.press/v37/ioffe15.html`.

[20] Espen Johnsen, Ronald Pedersen, and Egil Ona. Size-dependent frequency response of sandeel schools. *ICES Journal of Marine Science*, 66(6):1100–1105, 04 2009. ISSN 1054-3139. doi: 10.1093/icesjms/fsp091.
URL: `https://doi.org/10.1093/icesjms/fsp091`.

[21] Espen Johnsen, Guillaume Rieucau, Egil Ona, and Georg Skaret. Collective structures anchor massive schools of lesser sandeel to the seabed, increasing vulnerability to fishery. *Marine Ecology Progress Series*, 573:229–236, 2017.

[22] Espen Johnsen, Atle Totland, Åsmund Skålevik, Arne Johannes Holmin, Gjert Endre Dingsør, Edvin Fuglebakk, and Nils Olav Handegard. Stox: An open source software for marine survey analyses. *Methods in Ecology and Evolution*, 10(9):1523–1528, 2019.

[23] Espen Johnsen, Atle Totland, and Cecilie Kvamme. Measuring distribution and density of sprat in årdalsfjorden with a kayak drone-15-16 august 2020. *Rapport fra havforskningen*, 2020.

[24] RJ Korneliussen, E Ona, I Eliassen, Y Heggelund, R Patel, OR Godø, C Giertsen, D Patel, E Nornes, T Bekkvik, et al. The large scale survey system-lsss. In *Proceedings of the 29th Scandinavian Symposium on Physical Acoustics, Ustaoset*, volume 29, 2006.

[25] Rolf J Korneliussen. Acoustic target classification. 2018.

[26] Rolf J Korneliussen, Noel Diner, Egil Ona, Laurent Berger, and Paul G Fernandes. Proposals for the collection of multifrequency acoustic data. *ICES Journal of Marine Science*, 65(6):982–994, 2008.

[27] Rolf J Korneliussen, Yngve Heggelund, Gavin J Macaulay, Daniel Patel, Espen Johnsen, and Inge K Eliassen. Acoustic identification of marine species using a feature library. *Methods in Oceanography*, 17:187–205, 2016.

[28] Jan Kukačka, Vladimir Golkov, and Daniel Cremers. Regularization for deep learning: A taxonomy. *arXiv preprint arXiv:1710.10686*, 2017.

[29] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[30] S Mackinson, K Turner, D Righton, and JD Metcalfe. Using acoustics to investigate changes in efficiency of a sandeel dredge. *Fisheries Research*, 71(3):357–363, 2005.

[31] Ketil Malde, Nils Olav Handegard, Line Eikvil, and Arnt-Børre Salberg. Machine intelligence and the data-driven future of marine science. *ICES Journal of Marine Science*, 77(4):1274–1285, 2020.

[32] Kongsberg Maritime. Trans, 2022 (accessed May 9, 2022).
URL: `https://www.kongsberg.com/maritime/products/commercial-fisheries/td/`.

[33] Tunai Porto Marques, Melissa Cote, Alireza Rezvanifar, Alexandra Branzan Albu, Kaan Ersahin, Todd Mudge, and Stéphane Gauthier. Instance segmentation-based identification of pelagic species in acoustic backscatter data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4378–4387, 2021.

[34] Tunai Porto Marques, Alireza Rezvanifar, Melissa Cote, Alexandra Branzan Albu, Kaan Ersahin, Todd Mudge, and Stéphane Gauthier. Detecting marine species in echograms via traditional, hybrid, and deep learning frameworks. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 5928–5935. IEEE, 2021.

[35] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2, 2014.

[36] Alistair Miles. zarr-developers/zarr-python, April 2022.
URL: `https://github.com/zarr-developers/zarr-python`.

[37] Chandrahas Mishra and DL Gupta. Deep machine learning and neural networks: An overview. *IAES International Journal of Artificial Intelligence*, 6(2):66, 2017.

[38] Z Mohammed. *Acoustic identification of sandeel (Ammodytes marinus) using multi-frequency methods*. PhD thesis, MSc thesis, Department of Biology, University of Bergen, 2006.

[39] Alicia Mosteiro, Paul G Fernandes, F Armstrong, and SPR Greenstreet. A dual frequency algorithm for the identification of sandeel school echotraces. *ICES Document CM*, 12:1–13, 2004.

[40] Maryam M Najafabadi, Flavio Villanustre, Taghi M Khoshgoftaar, Naeem Seliya, Randall Wald, and Edin Muharemagic. Deep learning applications and challenges in big data analytics. *Journal of big data*, 2(1):1–21, 2015.

[41] Alba Ordonez Nils Olav Handegaard, Ibrahim Umar and Ingrid Utseth. Crimac pipeline - github, 2021 (accessed November 29, 2021).
URL: `https://github.com/CRIMAC-WP4-Machine-learning/CRIMAC-classifiers-unet`.

[42] Institute of Marine Research. Quota advice, 2021 (accessed September 23, 2021).
URL: `https://www.hi.no/en/hi/radgivning/quota-advice-1`.

[43] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

[44] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.
URL: `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[45] David MW Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. *arXiv preprint arXiv:2010.16061*, 2020.

[46] Rishav Kumar Rajak and Ashar Beg Mirza. Segmentation of polyp instruments using unet based deep learning model. 2021.

[47] Saman Razavi. Deep learning, explained: Fundamentals, explainability, and bridgeability to process-based modelling. *Environmental Modelling & Software*, 144:105159, 2021.

[48] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[49] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[50] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.
URL: `https://proceedings.neurips.cc/paper/2018/file/905056c1ac1dad141560467e0a99e1cf-Paper.pdf`.

[51] Ochin Sharma. A new activation function for deep neural network. In *2019 international conference on machine learning, big data, cloud and parallel computing (COMITCon)*, pages 84–86. IEEE, 2019.

[52] John Simmonds and David N MacLennan. *Fisheries acoustics: theory and practice.* John Wiley & Sons, 2008.

[53] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1139–1147, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.
URL: `https://proceedings.mlr.press/v28/sutskever13.html`.

[54] Guido Van Rossum. *The Python Library Reference, release 3.8.2.* Python Software Foundation, 2020.

[55] Ursula K. Verfuss, Ana Sofia Aniceto, Danielle V. Harris, Douglas Gillespie, Sophie Fielding, Guillermo Jiménez, Phil Johnston, Rachael R. Sinclair, Agnar Sivertsen, Stian A. Solbø, Rune Storvold, Martin Biuw, and Roy Wyatt. A review of unmanned vehicles for the detection and monitoring of marine fauna. *Marine Pollution Bulletin*, 140: 17–29, 2019. ISSN 0025-326X. doi: https://doi.org/10.1016/j.marpolbul.2019.01.009. **URL:** `https://www.sciencedirect.com/science/article/pii/S0025326X19300098`.

[56] Yu Weng, Tianbao Zhou, Yujie Li, and Xiaoyu Qiu. Nas-unet: Neural architecture search for medical image segmentation. *IEEE Access*, 7:44247–44257, 2019.

[57] Zhengxin Zhang, Qingjie Liu, and Yunhong Wang. Road extraction by deep residual u-net. *IEEE Geoscience and Remote Sensing Letters*, 15(5):749–753, 2018.

[58] Zongwei Zhou, Md Mahfuzur Rahman Siddiquee, Nima Tajbakhsh, and Jianming Liang. Unet++: A nested u-net architecture for medical image segmentation. In *Deep learning in medical image analysis and multimodal learning for clinical decision support*, pages 3–11. Springer, 2018.

Appendix A

Tools used

A.1 Windows Azure

Windows Azure [8] is a platform owned by Microsoft that provides cloud solutions for several services. It was used to access the remote storage provided by the IMR and mount this to a local computer. Thus enabling the downloading of the data for this thesis from IMRs server.

A.2 Docker

Docker [35] is an open-source platform that provides what they call containerization and is owned by the company under the same name, Docker, Inc. Docker is based on the Linux kernel, and enables you to create a container, which is an independent process that uses resources from the main instance, like virtual machine on a server but here applications. For each container, you can manage its own dependencies like programming languages and libraries. These containers can then be shared with others as images files, and as they can be run without the receiver having to manage the aforementioned dependencies as this is built into the image. Thus, you can make an application or code easily accessible for other people, as long as they have installed Docker. Docker was used to access and run the CRIMAC pipeline [41].

A.3 Zarr

By using the .zarr [36] format, you gain access to store chunked compressed multidimensional arrays. There are several highlights from this library, but was used primarily to access the arrays on disk. This means we did not need to load the entire array into memory and could work with the array and access parts of it without hardware limitations.

A.4 Xarray

Xarray[16] is a Python package that is made for working with multidimensional arrays. It is based on NumPy and adds labels in the form of attributes and coordinates on top of the NumPy-arrays. This was the library used for accessing and working more efficiently with the *.zarr* arrays, as this library has more functionality.

A.5 Pickle

To *pickle*[54] a file, means to use the built-in Python package pickle, which *serializes* Python object structures into byte streams. These can then be *unpickled* which means to *deserialize*, which is the opposite operation. As an example, this can be used to store and load Python arrays or machine learning models to and from disk.

Appendix B

Supplementary Results

B.1 All subsets in increasing order of mean F1-score

	Frequencies in test	Num	Precision	Recall	F1_Score
0	test_38kHz_18kHz_120kHz_200kHz	4	0.53	0.94	0.67
1	test_38kHz_18kHz_70kHz_120kHz_200kHz	5	0.52	0.94	0.67
2	test_38kHz_18kHz_120kHz_200kHz_333kHz	5	0.53	0.89	0.67
3	test_38kHz_18kHz_70kHz_120kHz_200kHz_333kHz	6	0.53	0.89	0.66
4	test_38kHz_18kHz_70kHz_200kHz_333kHz	5	0.53	0.89	0.66
5	test_38kHz_18kHz_70kHz_200kHz	4	0.51	0.93	0.66
6	test_38kHz_18kHz_200kHz	3	0.5	0.93	0.65
7	test_38kHz_18kHz_200kHz_333kHz	4	0.51	0.89	0.65
8	test_38kHz_18kHz_70kHz_120kHz_333kHz	5	0.44	0.86	0.58
9	test_38kHz_18kHz_120kHz_333kHz	4	0.44	0.86	0.58
10	test_18kHz_70kHz_120kHz_200kHz_333kHz	5	0.43	0.89	0.58
11	test_18kHz_70kHz_200kHz_333kHz	4	0.42	0.88	0.57
12	test_38kHz_18kHz_70kHz_333kHz	4	0.42	0.83	0.56
13	test_38kHz_18kHz_333kHz	3	0.43	0.82	0.56
14	test_18kHz_70kHz_200kHz	3	0.39	0.92	0.55
15	test_18kHz_120kHz_200kHz_333kHz	4	0.4	0.87	0.54
16	test_18kHz_70kHz_120kHz_200kHz	4	0.38	0.93	0.54
17	test_18kHz_200kHz_333kHz	3	0.4	0.84	0.53
18	test_18kHz_120kHz_333kHz	3	0.37	0.84	0.51

19	test_18kHz_120kHz_200kHz	3	0.36	0.9	0.51
20	test_18kHz_70kHz_333kHz	3	0.36	0.82	0.5
21	test_38kHz_70kHz_120kHz_200kHz_333kHz	5	0.38	0.7	0.5
22	test_18kHz_70kHz_120kHz_333kHz	4	0.35	0.86	0.49
23	test_70kHz_120kHz_200kHz_333kHz	4	0.39	0.66	0.49
24	test_38kHz_120kHz_200kHz_333kHz	4	0.37	0.68	0.48
25	test_70kHz_200kHz_333kHz	3	0.38	0.64	0.47
26	test_38kHz_200kHz_333kHz	3	0.37	0.65	0.47
27	test_38kHz_70kHz_200kHz_333kHz	4	0.36	0.68	0.47
28	test_38kHz_70kHz_120kHz_200kHz	4	0.35	0.68	0.46
29	test_70kHz_200kHz	2	0.36	0.64	0.46
30	test_18kHz_333kHz	2	0.34	0.73	0.46
31	test_70kHz_120kHz_200kHz	3	0.35	0.64	0.45
32	test_38kHz_70kHz_200kHz	3	0.34	0.68	0.45
33	test_18kHz_200kHz	2	0.3	0.84	0.45
34	test_38kHz_120kHz_200kHz	3	0.33	0.68	0.44
35	test_38kHz_18kHz_70kHz_120kHz	4	0.3	0.88	0.44
36	test_38kHz_200kHz	2	0.32	0.67	0.44
37	test_38kHz_333kHz	2	0.35	0.54	0.43
38	test_18kHz_70kHz_120kHz	3	0.28	0.86	0.42
39	test_38kHz_70kHz_120kHz_333kHz	4	0.31	0.63	0.42
40	test_120kHz_200kHz_333kHz	3	0.34	0.53	0.41
41	test_38kHz_70kHz_333kHz	3	0.31	0.62	0.41
42	test_38kHz_18kHz_120kHz	3	0.27	0.88	0.41
43	test_70kHz_120kHz_333kHz	3	0.32	0.56	0.4
44	test_70kHz_120kHz	2	0.3	0.62	0.4
45	test_38kHz_120kHz_333kHz	3	0.3	0.57	0.39
46	test_70kHz_333kHz	2	0.32	0.48	0.38
47	test_200kHz_333kHz	2	0.31	0.48	0.37
48	test_18kHz_120kHz	2	0.23	0.84	0.37
49	test_120kHz_200kHz	2	0.28	0.51	0.36
50	test_38kHz_70kHz_120kHz	3	0.25	0.64	0.36
51	test_120kHz_333kHz	2	0.29	0.44	0.35
52	test_18kHz_70kHz	2	0.23	0.78	0.35

53	test_200kHz	1	0.28	0.47	0.34
54	test_38kHz_18kHz_70kHz	3	0.22	0.81	0.34
55	test_38kHz_18kHz	2	0.22	0.76	0.34
56	test_38kHz_120kHz	2	0.23	0.64	0.34
57	test_70kHz	1	0.22	0.51	0.3
58	test_120kHz	1	0.21	0.48	0.29
59	test_38kHz_70kHz	2	0.19	0.57	0.28
60	test_18kHz	1	0.19	0.49	0.27
61	test_38kHz	1	0.18	0.5	0.26
62	test_333kHz	1	0.22	0.34	0.25

B.2 Tests per subset size in increasing order of mean F1-score

	Frequencies in test (1)	Precision	Recall	F1_Score
0	test_200kHz	0.28	0.47	0.34
1	test_70kHz	0.22	0.51	0.3
2	test_120kHz	0.21	0.48	0.29
3	test_18kHz	0.19	0.49	0.27
4	test_38kHz	0.18	0.5	0.26
5	test_333kHz	0.22	0.34	0.25

	Frequencies in test (2)	Precision	Recall	F1_Score
0	test_70kHz_200kHz	0.36	0.64	0.46
1	test_18kHz_333kHz	0.34	0.73	0.46
2	test_18kHz_200kHz	0.3	0.84	0.45
3	test_38kHz_200kHz	0.32	0.67	0.44
4	test_38kHz_333kHz	0.35	0.54	0.43
5	test_70kHz_120kHz	0.3	0.62	0.4
6	test_70kHz_333kHz	0.32	0.48	0.38
7	test_200kHz_333kHz	0.31	0.48	0.37

8	test_18kHz_120kHz	0.23	0.84	0.37
9	test_120kHz_200kHz	0.28	0.51	0.36
10	test_120kHz_333kHz	0.29	0.44	0.35
11	test_18kHz_70kHz	0.23	0.78	0.35
12	test_38kHz_18kHz	0.22	0.76	0.34
13	test_38kHz_120kHz	0.23	0.64	0.34
14	test_38kHz_70kHz	0.19	0.57	0.28

	Frequencies in test (3)	Precision	Recall	F1_Score
0	test_38kHz_18kHz_200kHz	0.5	0.93	0.65
1	test_38kHz_18kHz_333kHz	0.43	0.82	0.56
2	test_18kHz_70kHz_200kHz	0.39	0.92	0.55
3	test_18kHz_200kHz_333kHz	0.4	0.84	0.53
4	test_18kHz_120kHz_333kHz	0.37	0.84	0.51
5	test_18kHz_120kHz_200kHz	0.36	0.9	0.51
6	test_18kHz_70kHz_333kHz	0.36	0.82	0.5
7	test_70kHz_200kHz_333kHz	0.38	0.64	0.47
8	test_38kHz_200kHz_333kHz	0.37	0.65	0.47
9	test_70kHz_120kHz_200kHz	0.35	0.64	0.45
10	test_38kHz_70kHz_200kHz	0.34	0.68	0.45
11	test_38kHz_120kHz_200kHz	0.33	0.68	0.44
12	test_18kHz_70kHz_120kHz	0.28	0.86	0.42
13	test_120kHz_200kHz_333kHz	0.34	0.53	0.41
14	test_38kHz_70kHz_333kHz	0.31	0.62	0.41
15	test_38kHz_18kHz_120kHz	0.27	0.88	0.41
16	test_70kHz_120kHz_333kHz	0.32	0.56	0.4
17	test_38kHz_120kHz_333kHz	0.3	0.57	0.39
18	test_38kHz_70kHz_120kHz	0.25	0.64	0.36
19	test_38kHz_18kHz_70kHz	0.22	0.81	0.34

	Frequencies in test (4)	Precision	Recall	F1_Score
0	test_38kHz_18kHz_120kHz_200kHz	0.53	0.94	0.67
1	test_38kHz_18kHz_70kHz_200kHz	0.51	0.93	0.66

2	test_38kHz_18kHz_200kHz_333kHz	0.51	0.89	0.65
3	test_38kHz_18kHz_120kHz_333kHz	0.44	0.86	0.58
4	test_18kHz_70kHz_200kHz_333kHz	0.42	0.88	0.57
5	test_38kHz_18kHz_70kHz_333kHz	0.42	0.83	0.56
6	test_18kHz_120kHz_200kHz_333kHz	0.4	0.87	0.54
7	test_18kHz_70kHz_120kHz_200kHz	0.38	0.93	0.54
8	test_18kHz_70kHz_120kHz_333kHz	0.35	0.86	0.49
9	test_70kHz_120kHz_200kHz_333kHz	0.39	0.66	0.49
10	test_38kHz_120kHz_200kHz_333kHz	0.37	0.68	0.48
11	test_38kHz_70kHz_200kHz_333kHz	0.36	0.68	0.47
12	test_38kHz_70kHz_120kHz_200kHz	0.35	0.68	0.46
13	test_38kHz_18kHz_70kHz_120kHz	0.3	0.88	0.44
14	test_38kHz_70kHz_120kHz_333kHz	0.31	0.63	0.42

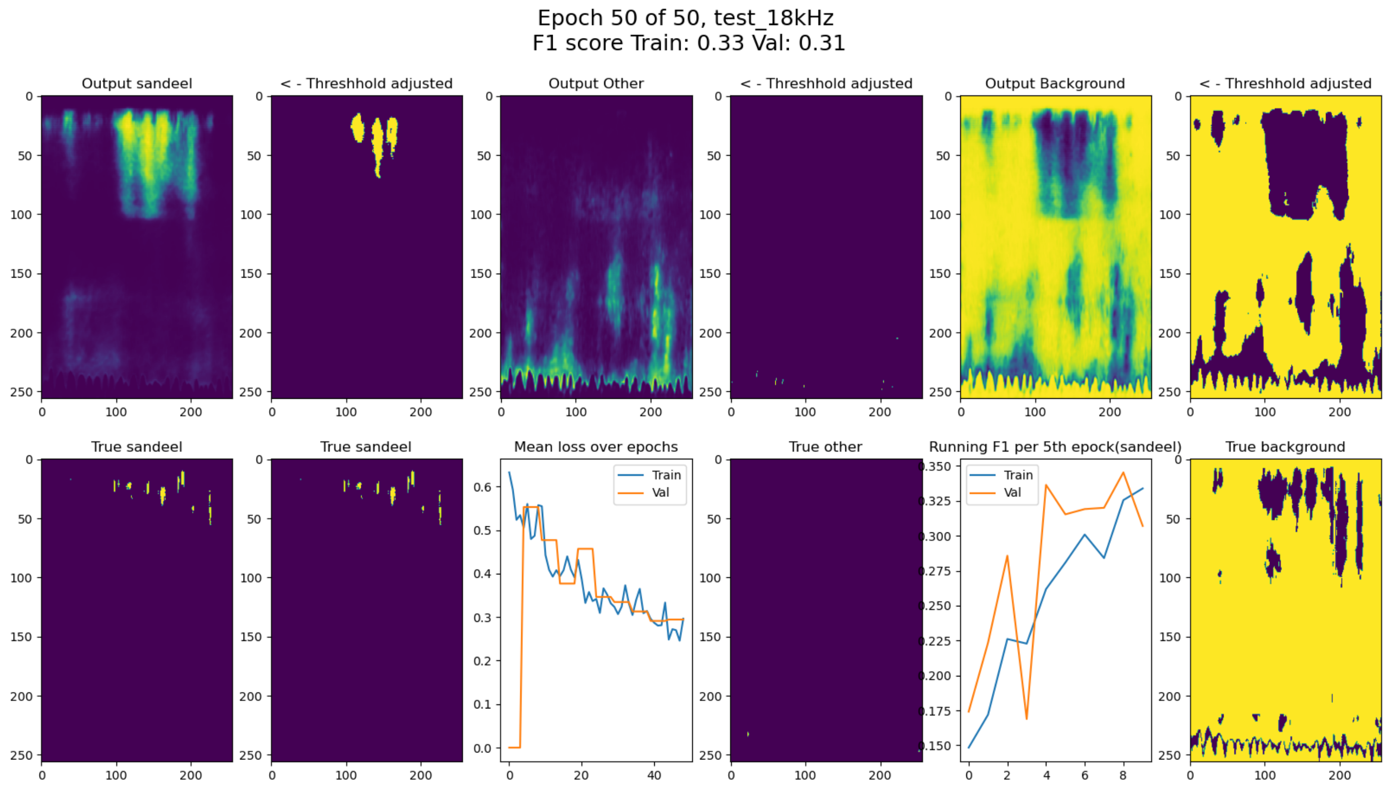
	Frequencies in test (5)	Precision	Recall	F1_Score
0	test_38kHz_18kHz_70kHz_120kHz_200kHz	0.52	0.94	0.67
1	test_38kHz_18kHz_120kHz_200kHz_333kHz	0.53	0.89	0.67
2	test_38kHz_18kHz_70kHz_200kHz_333kHz	0.53	0.89	0.66
3	test_38kHz_18kHz_70kHz_120kHz_333kHz	0.44	0.86	0.58
4	test_18kHz_70kHz_120kHz_200kHz_333kHz	0.43	0.89	0.58
5	test_38kHz_70kHz_120kHz_200kHz_333kHz	0.38	0.7	0.5

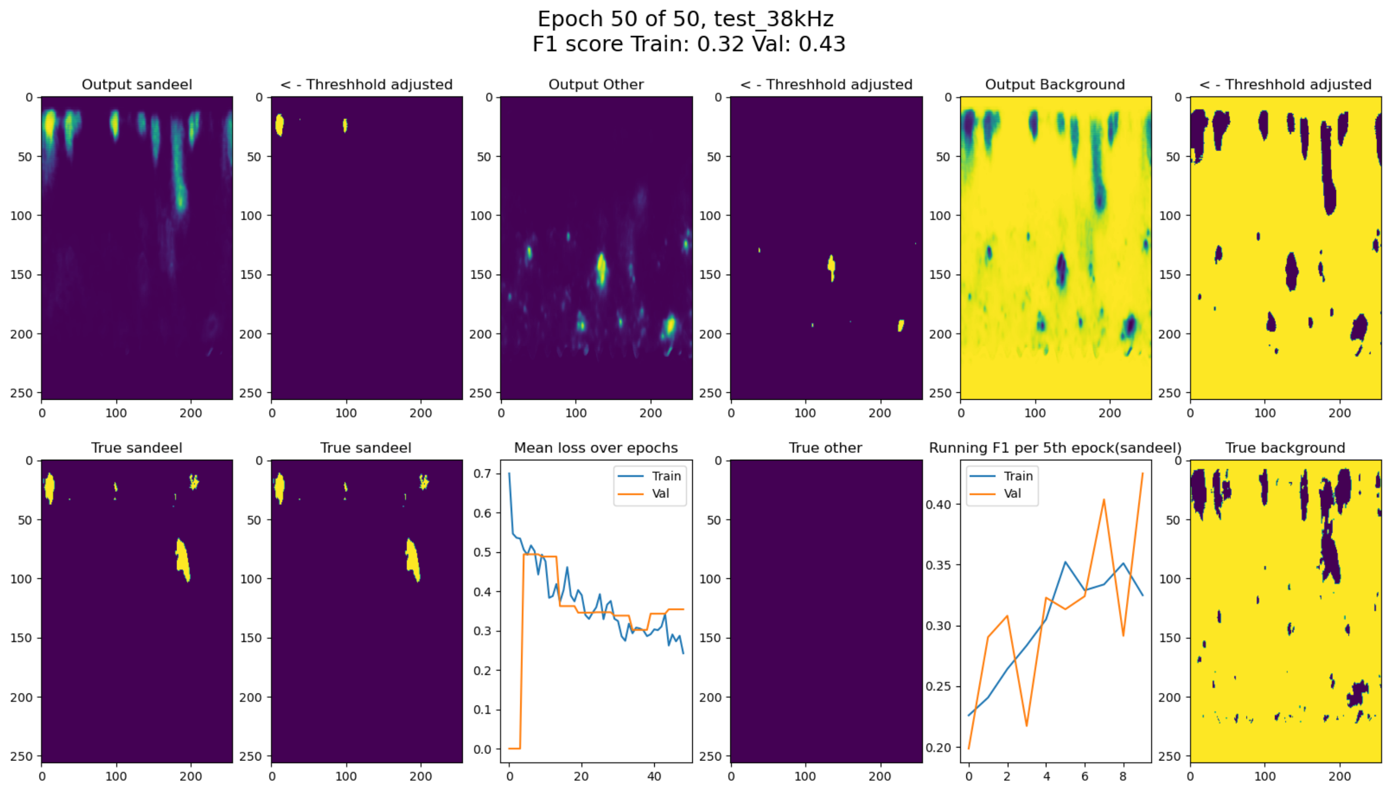
	Frequencies in test (6)	Precision	Recall	F1_Score
0	test_38kHz_18kHz_70kHz_120kHz_200kHz_333kHz	0.53	0.89	0.66

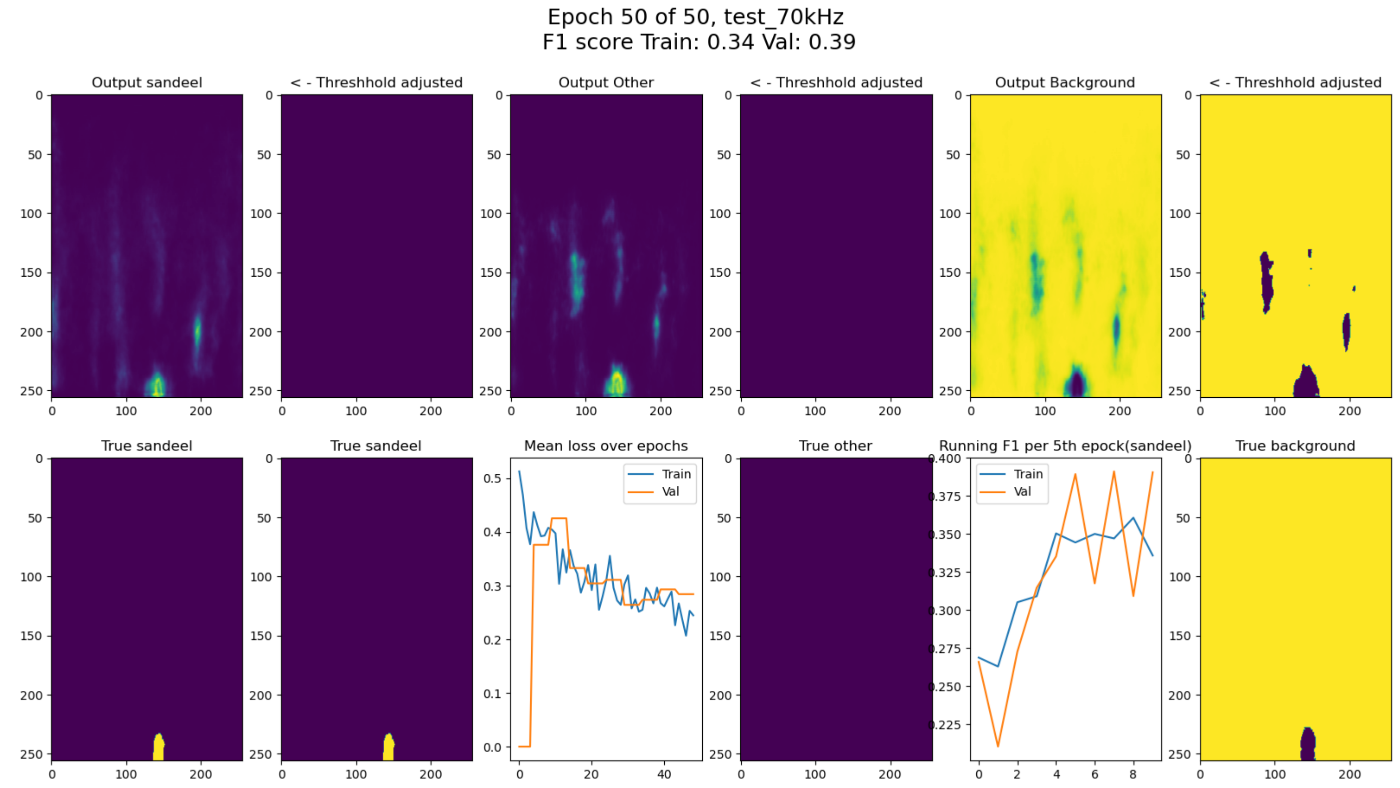
B.3 Training examples

This section includes more visualizations from the monitoring during training. At a subset size of one, we provide three illustrations from different tests, while only one plot from a common test is shown for increasing subsets up to, and including, five frequencies.

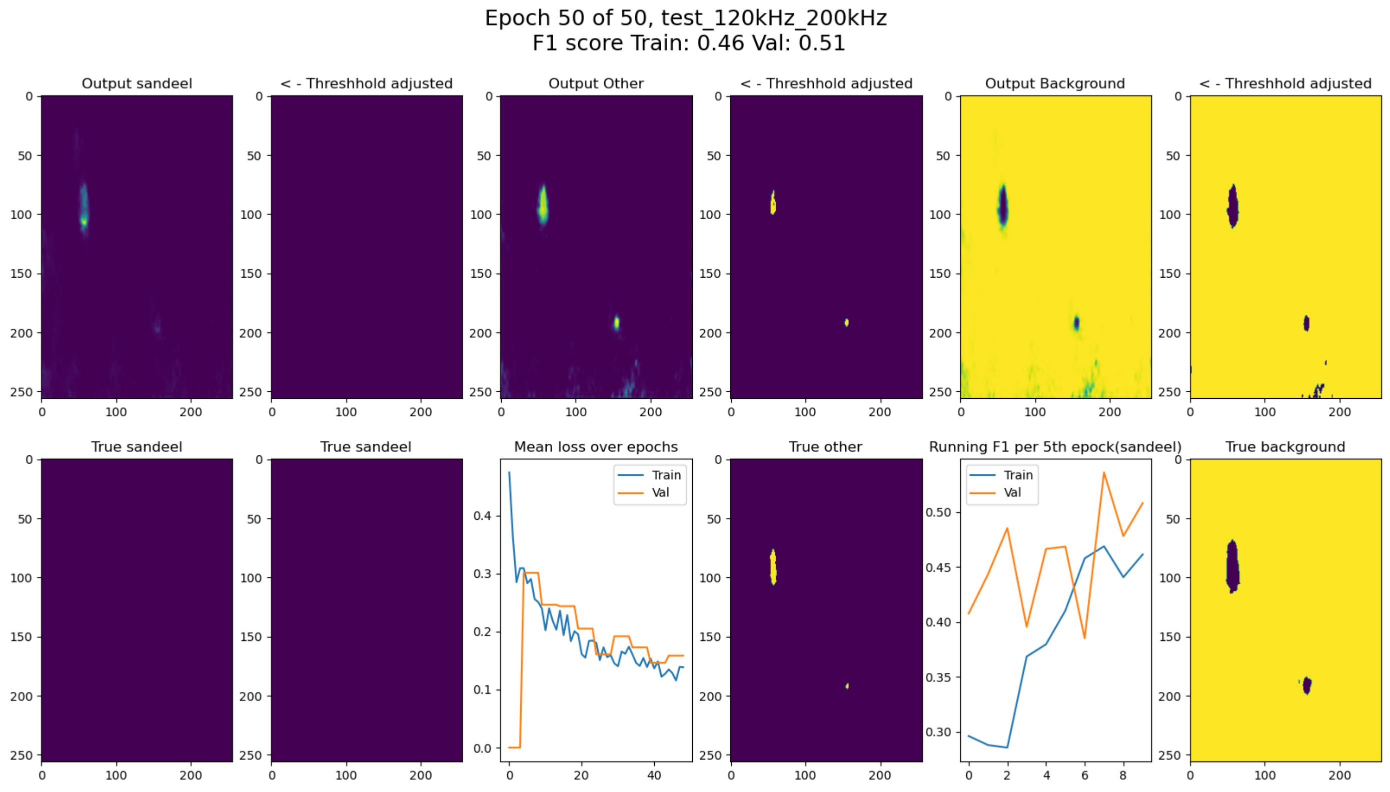
B.3.1 Subset size 1



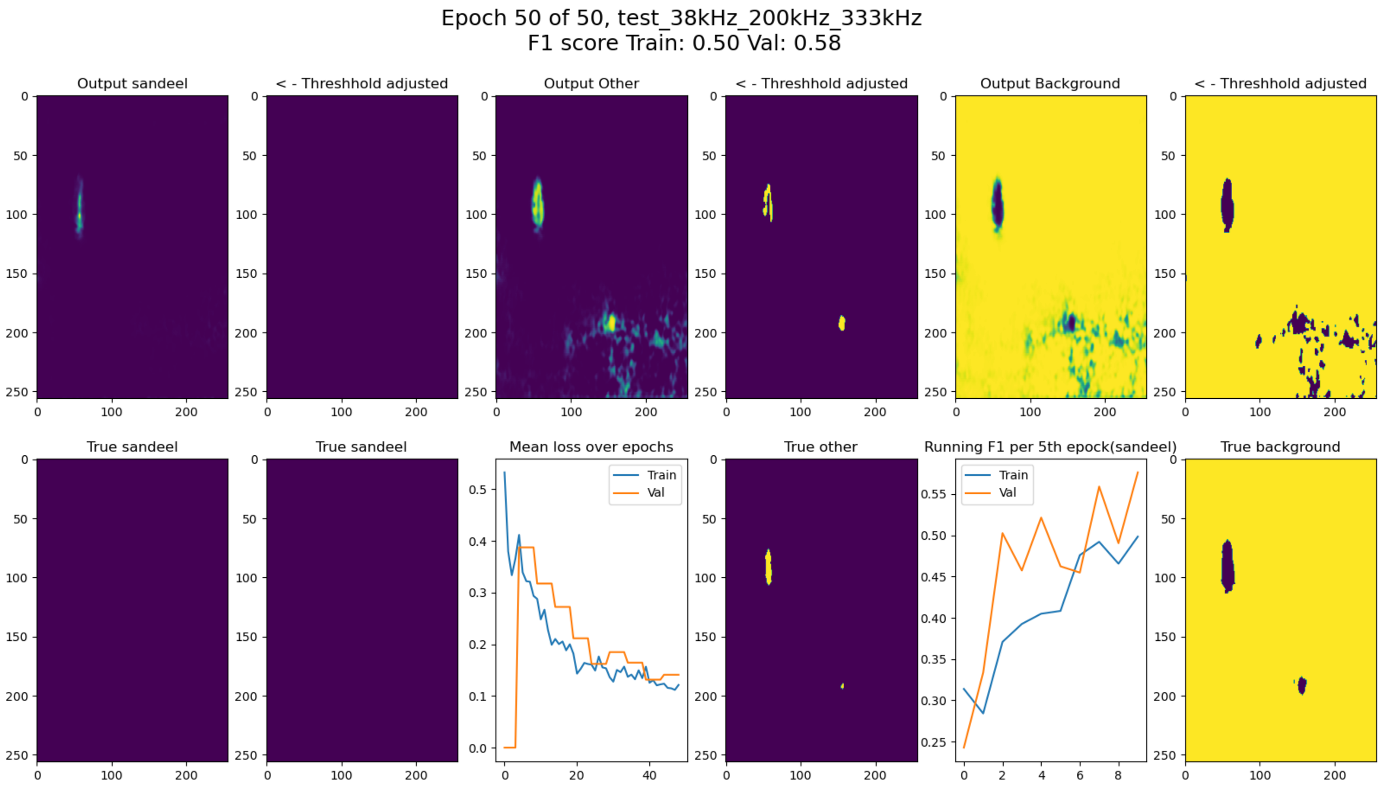

Epoch 50 of 50, test_38kHz
F1 score Train: 0.32 Val: 0.43
Output sandeel
< - Threshhold adjusted
Output Other
< - Threshhold adjusted
Output Background
< - Threshhold adjusted
True sandeel
True sandeel
Mean loss over epochs
Train
Val
True other
Running F1 per 5th epock(sandeel)
Train
Val
True background


Epoch 50 of 50, test_70kHz
F1 score Train: 0.34 Val: 0.39
Output sandeel
< - Threshhold adjusted
Output Other
< - Threshhold adjusted
Output Background
< - Threshhold adjusted
True sandeel
True sandeel
Mean loss over epochs
True other
Running F1 per 5th epock(sandeel)
True background
Train
Val

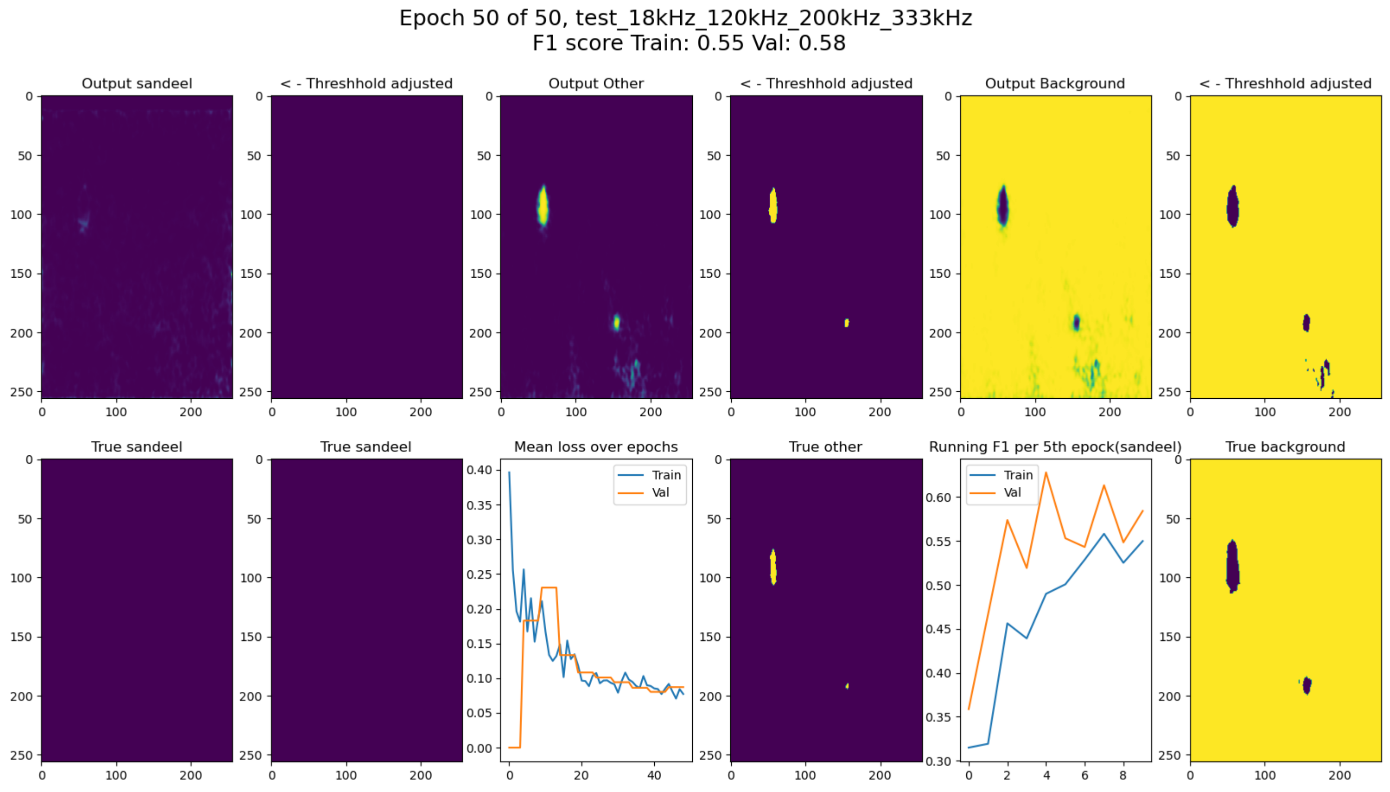
B.3.2 Subset size 2



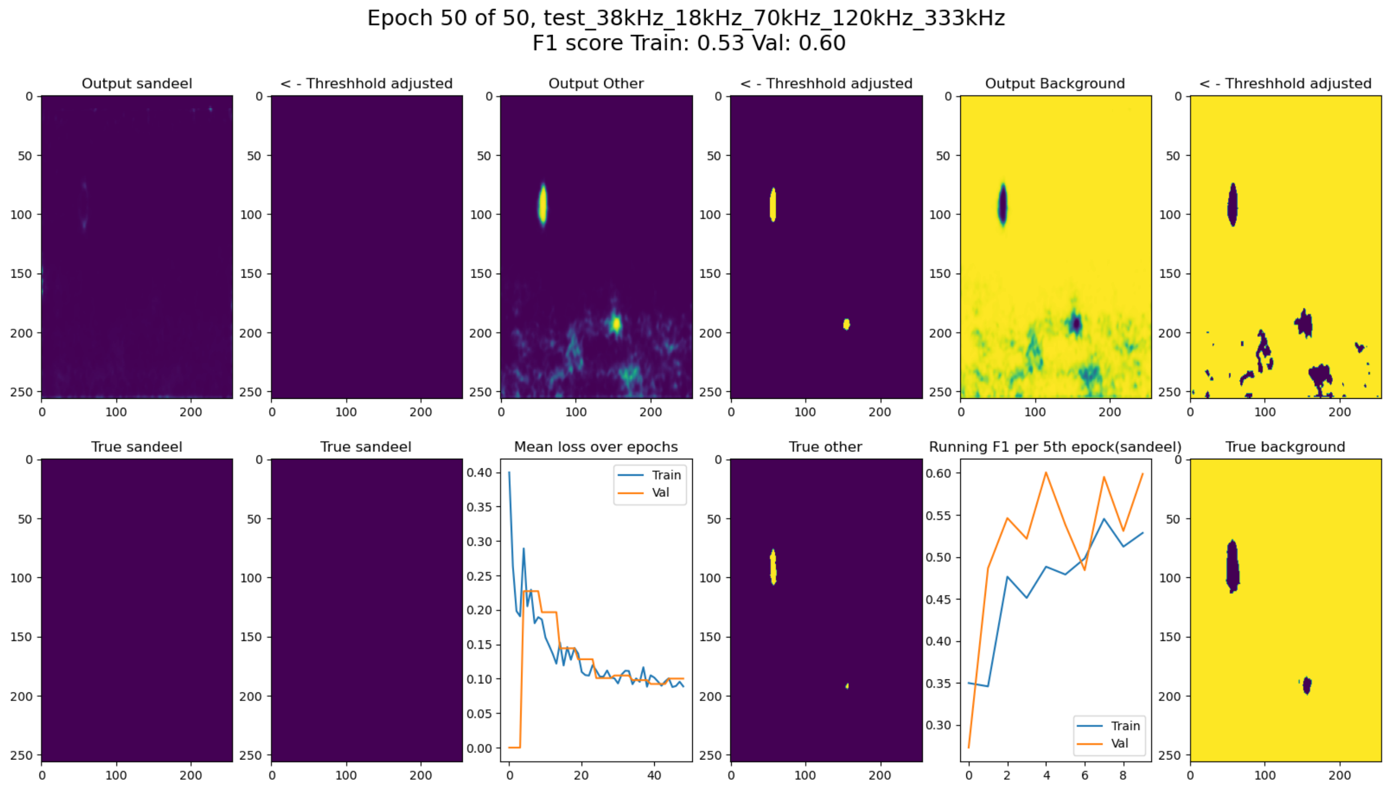
B.3.3 Subset size 3



B.3.4 Subset size 4



B.3.5 Subset size 5


Epoch 50 of 50, test_38kHz_18kHz_70kHz_120kHz_333kHz
F1 score Train: 0.53 Val: 0.60
Output sandeel
< - Threshhold adjusted
Output Other
< - Threshhold adjusted
Output Background
< - Threshhold adjusted
True sandeel
True sandeel
Mean loss over epochs
Train
Val
True other
Running F1 per 5th epock(sandeel)
Train
Val
True background