A Deep Learning Approach to Maritime Vessel Detection

University of Turku
Department of Computing
Master of Science Thesis
Computer Science
May 2024
Aleksi Kangas

Supervisors:
Luca Zelioli

UNIVERSITY OF TURKU
Department of Computing

Aleksi Kangas: A Deep Learning Approach to Maritime Vessel Detection

Master of Science Thesis, 63 p.
Computer Science
May 2024

The detection of maritime vessels is a fundamental task in maritime surveillance, and it is essential for various applications such as maritime traffic monitoring, search and rescue, and maritime security. Modern maritime surveillance systems rely on computer vision and deep learning techniques to detect and track maritime vessels in real-time. Maritime vessel detection from images is a challenging task due to various factors such as occlusions, varying illumination conditions, and long-range distances.

The focus of this thesis is on researching and experimenting with modern object detector architectures, backbone networks and different maritime datasets in order to understand the effects of different factors on the performance of maritime vessel detection. Using transfer learning, in total 6 different object detectors are trained and evaluated on 2 different maritime datasets. Used architectures include one-stage and two-stage object detectors. Experimentation is performed on consumer grade hardware.

The results of the experiment show that it is viable to develop a maritime vessel detection system using transfer learning and modern object detector architectures, even on consumer grade hardware. Quantitatively, the chosen one-stage architecture outperformed the chosen two-stage architecture with equivalent backbone networks, although the performance with both architectures were satisfactory. The qualitative results show that the major challenges in maritime vessel detection are related to environmental factors such as varying illumination conditions, long-range distances, and occlusions. Thus, additional research is needed to develop more robust maritime vessel detection systems. Techniques such as sensor fusion have been shown and could be used to improve the performance of maritime vessel detection systems, especially in challenging environmental conditions.

Keywords: vessel detection, object detection, maritime, transfer learning, computer vision

TURUN YLIOPISTO
Tietotekniikan laitos

Aleksi Kangas: A Deep Learning Approach to Maritime Vessel Detection

Pro gradu -tutkielma, 63 s.
Tietojenkäsittelytieteet
Toukokuu 2024

Meriliikenteen laivojen ja alusten havaitseminen on keskeinen haaste merivalvonnassa. Luotettava alusten havaitseminen on erityisen tärkeää erilaisissa sovelluskohteissa, kuten meriliikenteen seurannassa ja turvallisuudessa sekä etsintä- ja pelastustoimissa. Nykyaikaiset automaattiset merivalvontajärjestelmät hyödyntävät konenäköön ja syväoppimiseen perustuvia menetelmiä alusten havaitsemiseksi ja seuraamiseksi reaaliajassa. Meriliikenteen havaitseminen kuvista on haastavaa erilaisten tekijöiden, kuten vaihtelevien valaistusolosuhteiden, pitkien etäisyyksien ja esteiden vuoksi.

Tämän opinnäytetyön painopisteenä on nykyaikaisten kohteenhavaitsemisarkkitehtuurien, runkoneuroverkkojen ja erilaisten meriliikenteen tietoaineistojen tutkiminen ja kokeilu, ymmärtääksemme eri tekijöiden vaikutukset meriliikenteen alusten automaattiseen havaitsemiskykyyn. Siirto-oppimista hyödyntäen, yhteensä kuusi erilaista kohteenhavaitsemismallia koulutetaan ja arvioidaan kahdella erilaisella meriliikenteen tietoaineistolla. Käytetyt kohteenhavaitsemisarkkitehtuurit sisältävät yksi- ja kaksivaiheisia kohteenhavaitsemismalleja. Opinnäytetyön keskeinen koeasetelma suoritetaan kuluttajatason laitteistolla.

Kokeen tulokset osoittavat, että meriliikenteen alusten havaitsemisjärjestelmän kehittäminen on toteuttamiskelpoista, jopa kuluttajatason laitteistolla, hyödyntäen siirto-oppimista ja nykyaikaisia kohteenhavaitsemisarkkitehtuureja. Määrälliset tulokset osoittavat, että valittu yksivaiheinen arkkitehtuuri suoriutui paremmin kuin valittu kaksivaiheinen arkkitehtuuri vastaavilla runkoneuroverkoilla, vaikka suorituskyky molemmilla arkkitehtuureilla oli tyydyttävä. Laadulliset tulokset osoittavat, että suurimmat haasteet meriliikenteen alusten havaitsemisessa liittyvät ympäristötekijöihin, kuten vaihteleviin valaistusolosuhteisiin, pitkiin etäisyyksiin ja esteisiin. Näiden haasteiden vuoksi tarvitaan lisätutkimuksia kehittämään tehokkaita meriliikenteen laivojen ja alusten havaitsemisjärjestelmiä. Lisäksi erilaisia tekniikoita, kuten sensorifuusiota, on kirjallisuudessa ehdotettu hyödynnettäväksi havaitsemisjärjestelmien suorituskyvyn parantamiseksi, erityisesti haastavien ympäristöolosuhteiden vallitessa.

Asiasanat: aluksen havaitseminen, kohteen havaitseminen, merenkulku, siirto-oppiminen, konenäkö

Table of Contents

1 Introduction

1.1 Motivation

A deep understanding of the maritime environment is essential for a wide range of applications, including maritime traffic monitoring, surveillance and military operations. The maritime environment is complex and challenging, as it is affected by weather, lighting conditions and occlusions. Detecting vessels, i.e. what kind of ships or boats are present and where they are, is a crucial part of developing Situational Awareness (SA) in the maritime context. SA has been traditionally defined as a three-level process: perception of the environment, comprehension of the situation and prediction of the future state [1].

There are many ways to achieve *perception of the environment* in the maritime context, such as using radars, cameras, Automatic Identification System (AIS) and other sensors. For example, in [2] the authors have developed a novel detector based on Convolutional Neural Networks (CNNs) for multiscale Synthetic Aperture Radar (SAR) ship detection. In this thesis, the focus is on using traditional RGB cameras, i.e. using Computer Vision (CV) techniques to detect vessels from images.

Once the perception of the environment has been achieved, the next challenge is in understanding the situational context, which is crucial for the safety and efficiency of the operations, especially in port areas. *Comprehension of the situation* can be viewed as a traditional Pattern Recognition (PR) problem, where the goal is to

process the sensor information and uncover the patterns and regularities in the data. Techniques like object detection [3], image classification [4] and sensor fusion [5] are generally used to achieve comprehension of the situation. This thesis focuses on the comprehension of the situation by developing a vessel detection system using object detection techniques.

In practice, Artificial Intelligence (AI) and Machine Learning (ML) methods are the standard tools for developing *predictive models* for modern object detection. In vessel detection, Neural Networks (NN) and Deep Learning (DL) methods are widely used. For example, in [6], the authors have published a collection of papers on the topic of remote sensing in vessel detection and navigation, containing papers on subtopics such as *"Convolutional Neural Networks for Detection and Classification of Vessels"* and *"Object Detections by Different Sensors"*. Later maritime vessel detection research has also focused on Sensor Fusion (SF) techniques, such as fusing RGB and IR images [7] for better detections in challenging situations.

1.1.1 Problem Statement

For maritime traffic monitoring and safety purposes, it is essential to develop a robust vessel detection system that can detect vessels from images automatically. Such an automatic detection system would be very useful for port authorities, military organizations and other operators in monitoring and estimating the traffic flow both in ports and waterways and in open waters. This thesis aims to investigate the feasibility of developing a robust vessel detection system using publicly available data and models, and to expand the knowledge of the factors affecting the performance of the object detection in the maritime context.

1.1.2 Research Questions

The research questions of this thesis are centered around the feasibility of developing a robust vessel detection system using publicly available maritime datasets and pre-trained object detection models, which are run on consumer-grade hardware. More specifically, the thesis seeks to answer the following research questions:

1. How does the vessel detection performance differ between the two object detector architectures, one-stage SSD FPN (3.2.1) and two-stage Faster R-CNN (3.1.1)?

2. How does the performance of the object detectors differ when fine-tuned on the maritime datasets, ABOships (1.3.2 & 4.2.2) and SeaShips (1.3.3 & 4.2.3)?

3. How do different ResNet (3.3.1) feature extractors sizes affect the performance of SSD FPN and Faster R-CNN object detectors?

1.2 Literature Overview

This section briefly introduces the basic concepts of maritime situational awareness, pattern recognition, machine learning, transfer learning and vessel detection.

1.2.1 Maritime Situational Awareness

Situational Awareness (SA), defined as *"being aware of your surroundings"* [8], is a key concept in contexts like aviation, military and maritime. It is extremely important in military contexts to be able to identify and track vessels, but also in civilian contexts, such as maritime traffic monitoring and surveillance. Systems like Vessel Traffic Services (VTS) and River Information Services (RIS) serve as information providers for maritime SA in ports and waterways [9]. Van den Broek et al. [10] have presented a framework for maritime SA by combining sensor data

with intelligence and context information. Vessel detection based on object detection techniques can be used as part of the *observable* processing chain.

1.2.2 Pattern Recognition

Large amounts of data are being collected all around us, purchasing habits of customers, medical records of patients, images and videos from surveillance cameras, and much more. Pattern Recognition (PR) refers to a process of finding regularity structures and patterns in data. Moreover, Bishop [11, p. 1] defines PR as a field *"concerned with the automatic discovery of regularities in data through the use of computer algorithms and with the use of these regularities to take actions such as classifying the data into different categories"*. Figure 1.1 displays a simplified overview of the PR process.

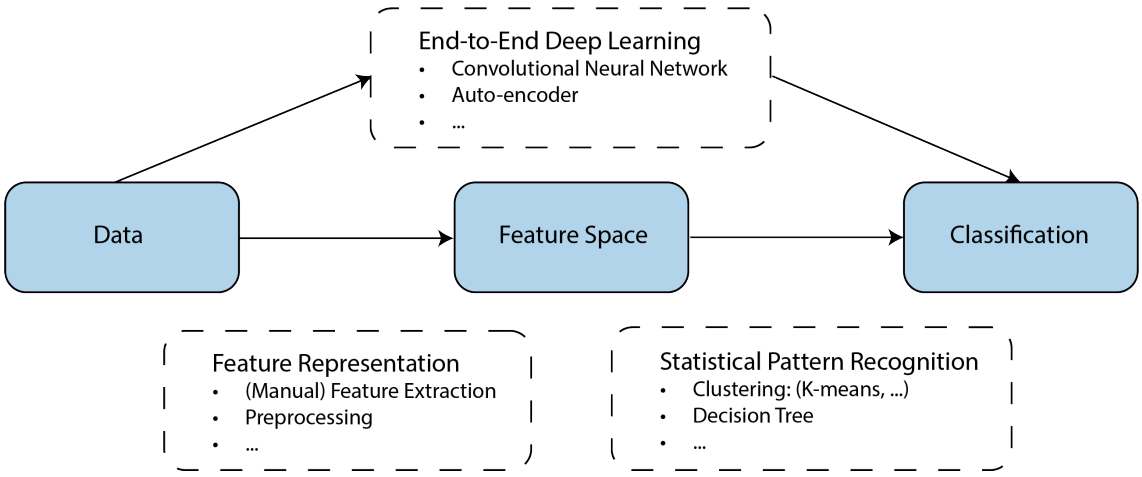

Figure 1.1: A simplified overview of the Pattern Recognition process. Notably, automated feature extraction via machine learning methods or even end-to-end deep learning have become extremely useful in practice, adapted from [12].

In Computer Vision (CV), PR tasks traditionally include image classification, object detection, image segmentation, image captioning, etc. The Convolutional Neural Network (CNN) architecture has been successfully used in many CV tasks, like image classification, semantic segmentation and importantly object detection [12], which is in the focus of this thesis.

Different PR problems could be solved by using hand-crafted rules and heuristics, i.e. by manual *feature extraction*, but such an approach is often infeasible and results in poor performance in practice [11]. Instead, Machine Learning provides a more general and automated approach to solving PR problems.

1.2.3 Machine Learning

Machine Learning (ML) allows computers to learn patterns from data without hand-crafted feature extraction. Deep Learning (DL) is a subfield of ML. Many common tasks in computing can be solved by using ML, such as regression (predicting a continuous value), classification (predicting a discrete value), machine translation, anomaly detection and much more. [13, p. 98-103]

ML methods can be grouped into unsupervised and supervised learning algorithms. In unsupervised learning, the algorithm learns patterns from unlabelled data. For example, clustering of data into groups of similar data points or learning the entire probability distribution of the data are unsupervised learning tasks. [13, p. 98-105]

In supervised ML algorithms, the algorithm learns patterns from labelled data, i.e. data where each sample is also associated with a label [13, p. 105-106]. Object (vessel) detection is a supervised learning task, where the algorithm learns to detect objects from images, using a set of images with known objects and their locations in the images.

A key challenge in ML is to develop a model that generalizes well on unseen data. Generally, we evaluate the performance of a model by measuring its performance on a dataset not used during training. The choice of the performance measure depends on the task at hand. [13, p. 104]

A common technique for assessing the performance of a model is with Cross-Validation (CV). CV is a *"data resampling method to assess the generalization abil-*

ity of predictive models and to prevent overfitting" [14]. In general, CV generates multiple training and testing sets from the original dataset, using random subsampling methods. Different variations of CV exist, such as the simple leave-one-out and (stratified) k-fold being the most commonly known variations. For example, in leave-one-out CV, only one sample is used for testing and the rest for training, and the splitting is applied for all samples, yielding a performance measure for each sample, which can be averaged to get the final performance measure. [14]

1.2.4 Transfer Learning in Object Detection

DL requires large amounts of labelled data for training, which is often not available in many domains. This is a problem in many applications, including maritime vessel detection. Additionally, training a DL model from scratch is computationally expensive. *Transfer Learning* (TL) is a ML technique that aims to improve performance in target domain by using the knowledge from a related source domain [15].

TL drastically reduces the amount of data needed for training a model. This is achieved by using a pre-trained model on a source domain, and then fine-tuning the model on the target task. [15] Practically, this means that the target model is initialized with the weights from an existing model, and then the model is trained on the target domain. Puttemans et al. [16] have shown that TL can be used to build robust industrial applicable object detection systems even with affordable hardware and modest amount of target domain data.

TL has been shown to be useful in vessel detection. The main challenge being the lack of huge vessel datasets, which is a problem shared by many other domain-specific object detection tasks. TL is an efficient way to use a trained network on another task, i.e. across domains [17].

Farahnakian et al. [17] have explored the performance of DL based vessel detection models using TL. Their results show that TL is important for training CNN-

based vessel detection models capable of maritime vessel detection.

1.2.5 Vessel Detection

It is challenging to develop a universal, automated and efficient CV-based vessel detection system due to the complexity of the maritime environment [18]. Different types of vessels, changing weather conditions and occasional occlusions hinder the development of such a system.

Dataset diversity is a key factor in developing a robust vessel detection system. For example, the SeaShips dataset (1.3.3) has been explicitly designed to contain images with different backgrounds, lighting environments, visible proportions of vessels, and occlusions. Even a large vessel can appear as a tiny object in the image if it is far away from the camera. Vessels can occlude each other, especially in crowded areas such as ports.

Wawrzyniak et al. [9] have proposed a detection method using video streams of existing monitoring system cameras. Even with the reasonably good results, the authors state that the incorrect detections are mainly due to unfavourable lighting conditions.

Another obvious challenge is night-time detection, where RGB images are not sufficient. Infra-Red (IR) cameras, and Sensor Fusion (SF) in general, allow for better detection in night-time conditions. For example, Farahnakian and Heikkonen [7] have explored different multi-modal fusion architectures utilizing RGB and IR images in vessel detection. Such techniques have been shown to improve the performance of vessel detection systems.

1.3 Datasets

Building a precise vessel detection model requires a large amount of maritime images. Images should contain vessels of different sizes and categories, depicted from various angles and distances, as well as in different weather and lighting conditions. A dataset is a collection of information (e.g. images, bounding boxes, labels) used for training and evaluating machine learning models.

1.3.1 COCO

The generic 'Microsoft Common Objects in Context' (COCO) dataset [19] contains 3 146 images of vessels. Originally, the COCO dataset was developed to advance the state-of-the-art in object recognition by gathering and annotating many images of common objects in their natural context. In total, the COCO dataset contains 328 000 images, 2.5 million labelled object instances and 91 object categories from everyday scenes. Examples of COCO images containing vessels are shown in Figure 1.2.

The COCO dataset has 3 146 images and 11 189 instances in the 'boat' category. It contains images of vessels of all sizes, from small rowboats to large ships. See Table 1.1 for the amount of images and objects, and the amount of vessel images and instances in the datasets. According to Iancu et al. [18], the COCO dataset has more vessel instances than other well-known general object detection datasets, like 'ImageNet' [20]. For maritime object detection, the COCO dataset is a reasonably good choice as the pre-training dataset, but for precise vessel detection, we need a proper maritime dataset. However, some specialized maritime datasets have been published in recent years, such as 'ABOships' and 'SeaShips'.

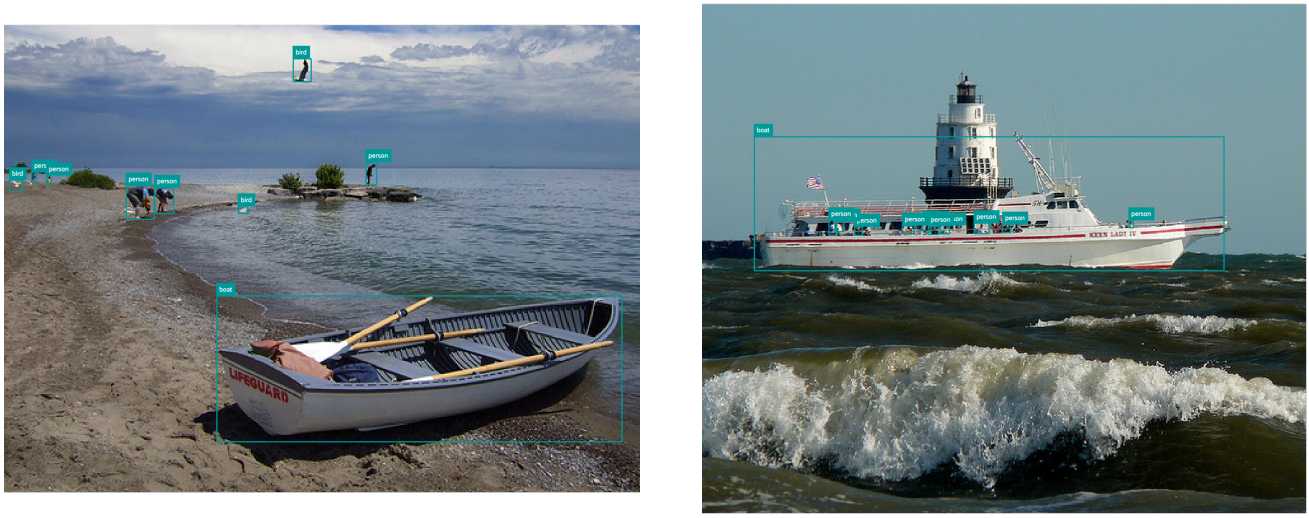

Figure 1.2: Examples of COCO images containing vessels of different sizes and types in the ‘boat‘ category.

1.3.2 ABOships

The ‘ABOships‘ dataset [18] is a new maritime dataset from 2021 consisting of inshore and offshore images of vessels from a watercraft traversing in the Turku (Åbo) region and the Finnish Archipelago. The dataset contains 9880 images of maritime objects (41967 in total) extracted from a collection of 720p 15 FPS videos at every 15 seconds (i.e. every 225 frames). Each annotated maritime object belongs to one of the 11 categories: ‘boat‘, ‘cargoship‘, ‘cruiseship‘, ‘ferry‘, ‘militaryship‘, ‘miscboat‘, ‘miscellaneous‘, ‘motorboat‘, ‘passengership‘, ‘sailboat‘ and ‘seamark‘. An example image is shown in Figure 1.3. The table 1.1 shows that ABOships dataset has the most categories, 11 in total, but two of them, ‘miscellaneous‘ and ‘seamark‘, are not vessels of any kind or have not been identified as such.

One of the advantages of the ABOships dataset is the amount of vessel objects as well as the information about the categories of vessels. The authors have focused on creating a maritime dataset by taking into account the challenges of maritime

Figure 1.3: An example image from the ABOships dataset. The dataset contains images of maritime objects from the Finnish Archipelago and the Turku (Åbo) region.

object detection, such as background variation, atmospheric conditions, occlusions and vessel scale variations [18]. Thus, the dataset is suitable for the purposes of investigating transfer learning capabilities of object detection models in maritime object detection.

1.3.3 SeaShips

The ‘SeaShips‘ dataset [21] is a large-scale maritime dataset from 2018. It consists of images of six common types of vessels: ‘ore carrier‘, ‘bulk cargo carrier‘, ‘general cargo ship‘, ‘container ship‘, ‘fishing boat‘ and ‘passenger ship‘ extracted from cameras of a coastline video surveillance system. The authors have focused on selecting images with different occlusions, backgrounds, vessel scales and other variations. The dataset contains 31 455 images of vessels (40 077 in total), which is more than the vessel images in the COCO dataset, and ABOships dataset. However, the publicly available subset [22] of the dataset contains only 7 000 images of the

total 31 455 images. An example image is shown in Figure 1.4. The table 1.1 shows that SeaShips dataset has the most vessel instances, but less vessel categories and in general less vessel instances per image than the ABOships dataset.

Figure 1.4: An example image from the SeaShips dataset. The dataset contains images of six common types of vessels from a coastline video surveillance system.

Compared to the ABOships dataset, the SeaShips dataset tends to have less vessel instances per image. This is because the SeaShips dataset has been collected from narrow port areas, whereas the ABOships dataset has been collected from both the Aura-river and the open waters of the Finnish Archipelago. On the other hand, the SeaShips dataset has vessel categories focused more on larger vessels, such as 'ore carrier' and 'bulk cargo carrier'. It should be taken into account when discussing about real world applications of the object detection models. The SeaShips dataset is suitable for the purposes of investigating the performance of the vessel detection models.

Dataset	**COCO**	**ABOships**	**SeaShips**
Images	328 000	9 880	31 455
Instances	2 500 000	41 967	40 077
Categories	91	11	6
Vessel Images	3 146	7 992	31 455
Vessel Instances	11 189	34 100	40 077
Vessel Categories	1	9	6

Table 1.1: Summary of the amount of images, instances and categories of the datasets.

1.4 Thesis Outline

This thesis is structured in a way, where the theoretical background is presented first, followed by the experiment and results. Chapter 1 has introduced the motivation and problem statement of the thesis, as well a brief literature overview of the central concepts, and the datasets used in the experiment. Chapter 2 introduces the basic concepts of (convolutional) neural networks and deep learning necessary for understanding the feature extractor (e.g. ResNet) and the basis of the object detection models. Chapter 3 introduces the theory of object detection, transfer learning and the two object detection models, SSD and Faster R-CNN, used in the experiment. Chapter 4 states the experiment setup in which the two pre-trained object detection models are trained and evaluated on the two maritime datasets, 'ABOships' and 'SeaShips'. Chapter 5 presents the results of the experiment and discussion about the results and future work. Chapter 6 summarizes and concludes the thesis.

2 Neural Networks

Neural Networks (NNs), inspired by the biological neurons of the brain, belong to the set of ML models. Artificial NNs consist of a network of many simple processing units, called *neurons*. Furthermore, the learning capabilities of NNs resemble the human brain by acquiring knowledge through a learning algorithm and storing the acquired knowledge into interneuron connection weights. NNs are described as massively parallel distributed processors and occasionally referred to as neurocomputers. [23, p. 24]

Humans have an inherent ability to transfer knowledge from one task to another. In contrast, traditional ML models are usually task-specific, meaning that they are only capable of solving the task they were trained for. *Transfer Learning* (TL) is a ML technique that aims to transfer learned knowledge from one task to another [24]. NNs include a capability to adapt the weights to the task at hand. A NN trained for one task environment can be easily adapted to another task environment by retraining it with a new set of training data [23, p. 25]. The focus of the thesis is on the TL capabilities of NNs in the context of vessel detection.

2.1 Neural Network Architecture

2.1.1 Neuron

Neuron is a fundamental unit of a NN. Highly inspired by the biological neuron of the human brain, it is a simple processing unit that takes a number of inputs, performs a computation and outputs a result. Mathematically, a neuron can be described as a function that takes a vector of inputs $\mathbf{x} = [x_1, x_2, ..., x_n]$ and outputs a scalar value y. The inputs $\mathbf{x}$ are both multiplied by learnable weights $\mathbf{w} = [w_1, w_2, ..., w_n]$ and added together to form the *logit* of the neuron, $z = \sum_{i=1}^{n} w_i x_i$. Frequently, a constant bias term b is added to the logit. The output of the neuron is then computed by applying a function f to the logit, yielding [25, p. 45-47]:

$$y = f(z) = f(\mathbf{x} \cdot \mathbf{w} + b). \tag{2.1}$$

The vector representation is crucial for the implementation of NNs, as it allows for efficient computation using matrix operations [25, p. 45-47]. A neuron is illustrated in Figure 2.1.

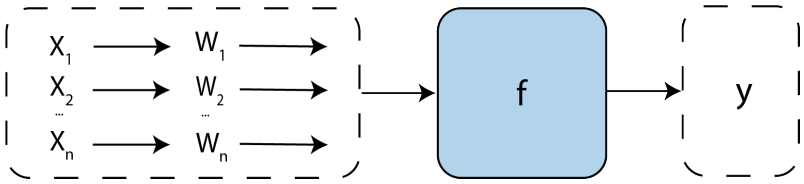

Figure 2.1: Illustration of a neuron, with inputs $x_1, x_2, ..., x_n$, weights $w_1, w_2, ..., w_n$, activation function f and output y.

The function f in Equation 2.1 is referred to as the *activation* function of the neuron. The original perceptron model developed by Rosenblatt in 1958 used a step function as the activation function [26]. Several other activation functions have been proposed since then, and the choice of activation function is an important design decision when building a NN. Three commonly used non-linear activation functions

are *sigmoid*, *softmax* and *rectified linear unit (ReLU)* [25, p. 51-54].

Sigmoid

The sigmoid function is defined as $f(z) = \frac{1}{1+e^{-z}}$. The S-shaped curvature of the function ensures that the output for a small logit is close to 0 and for a large logit close to 1. [25, p. 51] The output of the sigmoid is saturated for small and large logit values, meaning that the gradient of the function is close to 0. This introduces the *vanishing gradient* problem, where the gradient descent algorithm converges slowly or not at all. [27]

Softmax

The softmax function is defined as $f(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$. Softmax is essentially a generalization of the sigmoid function to multiple dimensions. When we have a discrete variable with n possible values and we want to represent a probability distribution over it, we can use the softmax function. The softmax function is commonly used as the output of a classifier. It is also susceptible to the saturation problem, caused by the input being extremely negative or positive. [13, p. 184-186]

ReLU

The ReLU (Rectified Linear Unit) [25, p. 52] function is defined as $f(z) = \max(0, z)$. It became a popular activation function after it was successfully used in the AlexNet architecture by Krizhevsky et al. at the ImageNet Large Scale Visual Recognition Challenge in 2012. They achieved the winning top-5 test error rate of 15.3% beating the second-best result 26,2% by using a deep convolutional NN with ReLU activation functions. The advantage of the ReLU is that it is non-saturating, which means that the vanishing gradient problem is alleviated. [28]

2.1.2 Feed-Forward Neural Networks

The idea of a NN is to connect multiple neurons together in a way that allows complex computations. The first NNs date back to the 1800s, but the basis of modern NNs was laid roughly in the 1950s and 1960s [29]. Rosenblatt proposed in 1958 [26] and 1962 [30] the concept of *perceptron* model and the general architecture of a *multi-layer perceptron*. Fundamentally, a single neuron has more expressive power than a linear perceptron, as it is capable of performing non-linear computations, provided that the activation function is non-linear. However, even a single neuron with a non-linear activation function is not capable of solving complex problems. [25, p. 47-51]

A modern NN is a collection of neurons organized into layers. In a NN, each neuron is connected to at least one other neuron, with the connections between the neurons being expressed with a numerical *weight* coefficient. A multi-layer feed-forward NN contains an *input* layer, an *output* layer, and one or more *hidden* layers in between. In a *fully-connected feed-forward* NN, every neuron within a layer is linked to every neuron in the subsequent layer of the network. [31]

Thus, the neurons within a layer of a fully-connected feed-forward neural network can be rearranged arbitrarily without affecting the final output of the network [25, p. 97]. Feed-forward NNs only contain forward connections, i.e. no *recurrent* connections or loops [25, p. 49, 208]. A typical fully-connected feed-forward NN is illustrated in Figure 2.2.

The input layer is the first layer of the NN, and in charge of taking the input of the NN in a vectorized form. For example, the input values could be the pixel values of an image, in which case the number of input neurons would equal the pixels in the image [25, p. 50].

The last layer of the NN is the output layer, which is responsible for outputting the result in a form that is suitable for the task at hand. [25, p. 50] For instance, a

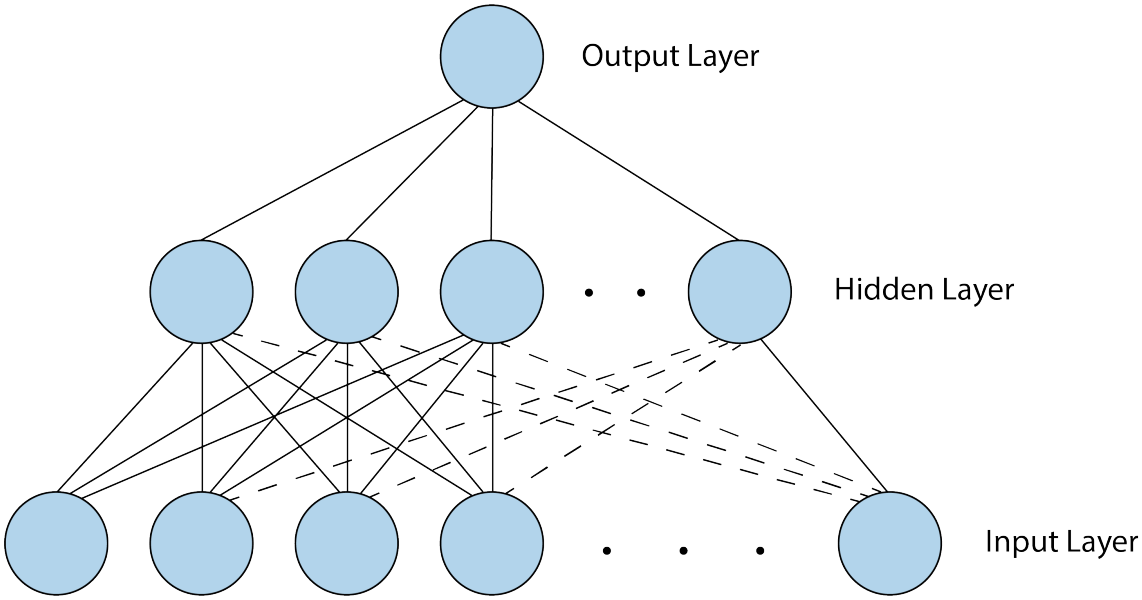

Figure 2.2: The structure of a typical fully-connected feed-forward neural network. Illustrated is only a single hidden layer, but the network may contain any number of hidden layers.

classification task with k classes would have k output neurons, where each neuron represents the probability of the input being of the corresponding class.

The rest of the layers between the input and output layers are commonly referred to as *hidden* layers [25]. The hidden layers are the basis of the expressive power of NNs, since they allow the NN to learn complex non-linear relationships between the input and output. Theoretical results suggest that a single hidden layer NN can represent any function, given a sufficiently large number of neurons [32].

In theory, hidden layers may contain any number of neurons. However, the number of neurons in each hidden layer is usually lower than the number of input neurons, allowing the NN to learn a compressed representation of the input data [25, p. 50]. For the NN to be able to solve non-linear problems (e.g. the simple *exclusive-or* (XOR) problem), it must contain at least one hidden layer using non-linear activation functions [13, p. 172]. Activation functions have been discussed in more detail in Section 2.1.1.

2.1.3 Deep Neural Networks

Formally, feed-forward NNs can be described as collections of functions f connected in a chain-like manner. For example, we can represent a NN of 3 layers with functions $f^{(1)}, f^{(2)}, f^{(3)}$ (where $f^{(i)}$ denotes the ith layer) chained together to yield $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$. The length of the function chain, i.e. the amount of layers, is referred to as the *depth* of the network. [13, p. 168] Figure 2.2 illustrates a feed-forward NN with 3 layers.

A *Deep Neural Networks* (DNNs) stacks many hidden layers on top of each other [32, p. 275]. *Deep Learning* (DL) refers to the class of ML models that utilize many-layered neural networks. In contrast to traditional ML methods, the benefit of DL is *end-to-end training*, which means that the entire model is trained at once, including *automatic* feature extraction and engineering, which are usually done manually in traditional ML methods. [32, p. 27-28]

DL has become popular during the recent decades, achieving breakthrough performance in many tasks, such as speech recognition, face detection and image classification. Traditionally, object detection has been performed with hand-crafted features and shallow trainable architectures. DL methods enable learning of semantic, high-level and deeper features from the data. These abilities have been shown to improve the performance in object detection. DL object detection methods have been and are being actively researched. [33]

2.2 Training Neural Networks

NNs are usually trained with gradient-based optimization algorithms, which iteratively update the weights of the network to minimize a *loss function*. The general approach to training of NNs does not differ much from the training of traditional ML models. [13, p. 177] The basic concepts of training NNs and relevant intricacies

are discussed in this section.

2.2.1 Optimization Problem

Optimization means either minimizing or maximizing a function $f(x)$ by choosing a suitable value for x. Learning from data, i.e. fitting a model to the data, is a fundamental optimization problem in ML. Most commonly, we tend to minimize a loss function. [13, p. 82]

Let $\mathbf{w}$ be the weights of the NN and b be the bias. The goal of the optimization problem is to find parameters $\boldsymbol{\theta} = (\mathbf{w}^*, b^*)$ which yield the minimal total loss, formally [32, p. 85]:

$$\mathbf{w}^*, b^* = \underset{\mathbf{w},b}{\operatorname{argmin}} L(\mathbf{w}, b). \tag{2.2}$$

Loss Function

For measuring how the NN performs, a *loss function* is used. Traditionally, the *loss* is a non-negative real value where smaller values indicate a better fit [32, p. 84]. Different loss functions are used for different tasks [34].

A common loss function for classification tasks is the *cross-entropy loss*. For any given example y and prediction $\hat{y}$, the cross-entropy loss is defined as [32, p. 131]:

$$L(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{j=1}^{q} y_j \log \hat{y}_j \tag{2.3}$$

Cross-entropy loss can be thought to both maximize the likelihood of the observed data and to minimize the information (number of bits) required for communicating the labels [32, p. 134]. The derivation of cross-entropy loss is based on *information theory* and the proof is omitted here. More about relevant information theory can be found in for example [35].

A common loss function for regression problems is the *smooth L1 loss*, defined

as [36]:

$$L(\mathbf{y}, \hat{\mathbf{y}}) = \sum_i \begin{cases} 0.5(\hat{y}^{(i)} - y^{(i)})^2 & \text{if } |\hat{y}^{(i)} - y^{(i)}| < 1 \\ |\hat{y}^{(i)} - y^{(i)}| - 0.5 & \text{otherwise} \end{cases} \tag{2.4}$$

where $\hat{y}^{(i)}$ is the prediction and $y^{(i)}$ is the true value for example i.

Back-Propagation

Optimization problems are commonly tackled with *gradient descent* algorithms. These algorithms rely on the *gradient* of the loss function $\nabla_{\mathbf{w},b} L(\mathbf{w}, b)$ to establish how to change the parameters to minimize the loss function L, i.e. traverse in the direction of the steepest descent. Mathematically, the gradient of a function f is defined as a vector of partial derivatives $\frac{\partial}{\partial x_i} f(\mathbf{x})$ which measure the change in f with respect to each of the input variables x_i. [13, p. 82-86]

Feed-forward NNs take an input $\mathbf{x}$ and compute an output $\hat{\mathbf{y}}$. The input $\mathbf{x}$ is transferred forward in the network until the output $\hat{\mathbf{y}}$ is computed along with a scalar loss value $L(\mathbf{w}, b)$. This step is called *forward propagation.* [13, p. 204]

In contrast, *back-propagation* allows *"the information from the cost to then flow backwards through the network, in order to compute the gradient"* with respect to the parameters of the network. The optimization algorithm uses the computed gradient value to update the parameters of the network. [13, p. 204]

The back-propagation algorithm relies on the chain rule of calculus. Formally, let $y = f(g(x))$. Additionally, let the functions $y = f(u)$ and $u = g(x)$ be differentiable. The chain rule is [32, p. 59]:

$$\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx}. \tag{2.5}$$

The chain rule allows to mathematically determine the gradient of the loss function, according to each parameter of the network. Suppose a chain $x = f(w), y =$

$f(x), z = f(y)$ of functions, where w is the parameter of interest. To compute $\frac{\partial z}{\partial w}$, we can use the chain rule as follows [13, p. 211]:

$$\frac{\partial z}{\partial w} = \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial x} \cdot \frac{\partial x}{\partial w} \tag{2.6}$$

2.2.2 Optimizers

When the gradient of the loss function is zero, the algorithm has reached a *critical point*, which is either a *local minimum*, *local maximum* or a *saddle point*. A critical point which yields the lowest value of the loss function is called the *global minimum*. Optimization in the field of DL is challenging, because the loss functions may have many local minima. [13, p. 82-86] The challenges in gradient-based optimization are illustrated in Figure 2.3.

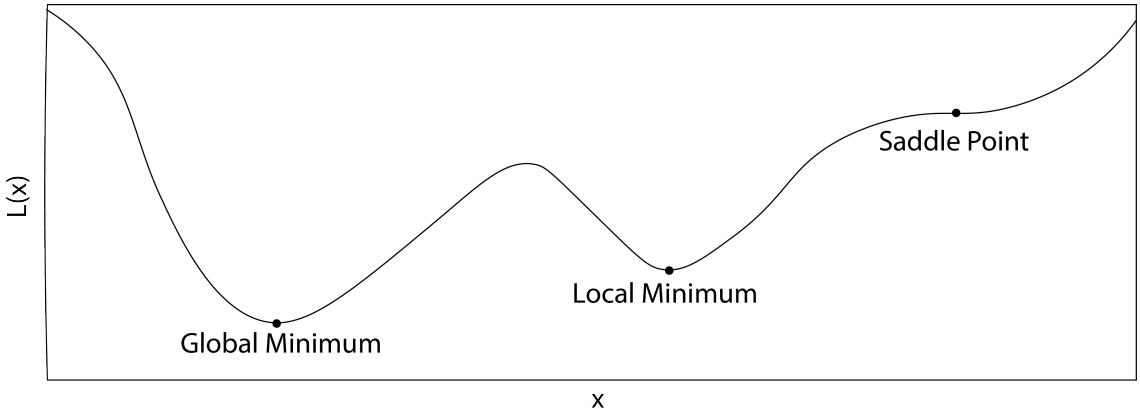

Figure 2.3: Challenges in gradient based optimization are many. Saddle points and poor local minima are obstacles when trying to find the global minimum.

Instead of finding the global minimum, we settle to finding a local minimum with a sufficiently low loss value [13, p. 82-86]. Many different optimizers based on gradient descent have been proposed ([37], [38], [39], [40], ...), and a few are presented in this section.

Stochastic Gradient Descent

Traditionally, the whole dataset is used to compute the gradient value of the loss function. This obviously does not scale well to large datasets. Since the gradient is an expectation, we can approximate it by using a small subset of the training data, a *minibatch* $\mathbb{B} = \{\mathbf{x}^{(1)}, ..., \mathbf{x}^{(m')}\}$. The estimate of the gradient is then [13, p. 152]:

$$\mathbf{g} = \frac{1}{m'} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m'} L(\mathbf{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}). \tag{2.7}$$

And the *stochastic gradient descent* (SGD) update rule is [13, p. 152]:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \mathbf{g} \tag{2.8}$$

where ϵ is the learning rate.

SGD is a popular optimization algorithm for neural networks, because it scales well to large datasets [13, p. 153]. Despite the popularity of SGD, it has some drawbacks. Training with SGD may be slow because of noisy and small gradients introducing variance to the learning process [13, p. 296-297].

Momentum Optimization

Momentum optimization, originally presented by Polyak in 1964 [37], is a modification to the SGD algorithm that accelerates the convergence. It relies on the concept of *momentum*, which in physics is defined as the product of mass and velocity. In the context of optimization, a unit mass is assumed, and the momentum is defined as the *velocity* that accumulates a moving average of the previous gradients, which is decayed exponentially. This alleviates the problem of variance caused by noisy gradients in SGD. [13, p. 296]

Formally, let velocity be $\mathbf{v}$ and α to be a hyperparameter for controlling exponential decay of the previous gradients. Then the momentum optimization algorithm

is defined as: [13, p. 296]

$$\begin{aligned} \mathbf{v} &\leftarrow \alpha\mathbf{v} - \epsilon\mathbf{g} \\ \boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} + \mathbf{v} \end{aligned} \tag{2.9}$$

2.3 Convolutional Neural Networks

In computer vision tasks, feature extraction has always been an important research topic. Traditionally, feature extraction has been based on pre-designed features derived from statistical regularities or prior knowledge [41]. While DL allows for automatic feature extraction, it is not feasible to use fully-connected NNs for image data, because of the huge amount of parameters needed.

A neuron in the hidden layer of a fully-connected NN would have 784 parameters when a 28×28 black-white image is used as input. With larger images, e.g. 200×200 image with 3 colour channels, the number of parameters in the input layer alone would be $200 \times 200 \times 3 = 120,000$. [25, p. 121-122]

Clearly, this is not a feasible approach. Inspired by human vision, Convolutional Neural Networks (CNNs) reduce the number of parameters drastically with an architecture suited for image data. [25, p. 121-122] CNNs provide an end-to-end learning framework for feature extraction [41].

2.3.1 Convolution

The core of CNNs is the *convolution* operation, where a *filter* or *kernel* is multiplied over the entire area of an input image [25, p. 123]. Given two real-valued functions x and w, the convolution operation is defined as $s(t) = (x * w)(t)$ [13, p. 331]. Since images are two-dimensional, two-dimensional convolution kernel is used and

the convolution operation is defined as [13, p. 332]:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{2.10}$$

where I is the input image, K is the kernel and S is the output. An example of a convolution operation is shown in Figure 2.4.

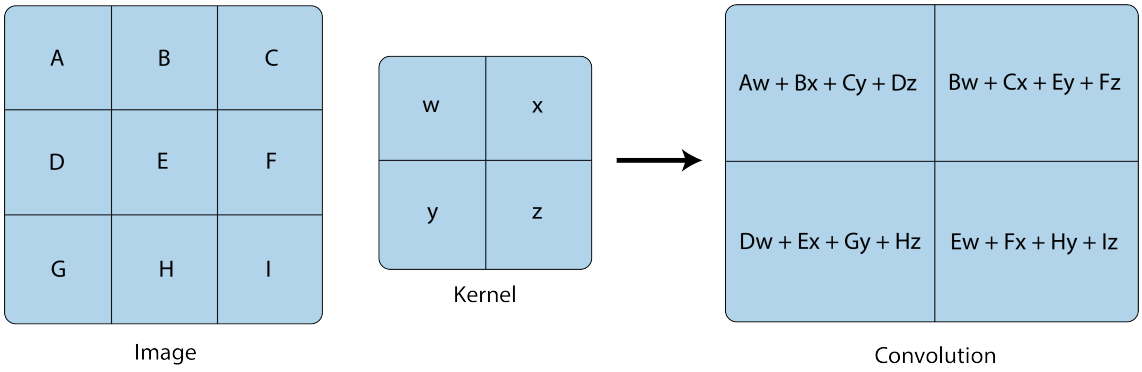

Figure 2.4: Simplified example of a convolution operation. The input image is 3×3 and the kernel is 2×2. The kernel is applied to only the valid positions without padding.

Convolution being commutative, the kernel can be flipped, and the convolution operation can be written as [13, p. 332]:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n) \tag{2.11}$$

which is referred to as *cross-correlation*, and is the operation used in practice for implementation reasons [13, p. 333].

If the input image shape is $n_h \times n_w$ and the kernel is $k_h \times k_w$, the shape of the convolution output is $(n_h - k_h + 1) \times (n_w - k_w + 1)$ [32, p. 254]. Thus, the output shape is smaller than the input shape, assuming that the kernel is larger than 1×1. Two techniques, *padding* and *stride*, are used to control the output shape of the convolution.

Padding

When applying convolution, by default the kernel is centred at each pixel of the input image [32, p. 247]. Pixels at the edges of the input image are used less than the pixels at the centre of the input image. Successive convolution operations obliterate the information at the edges of the image due to the reduction in the image size. To preserve the output shape, the effective size of the input image can be increased by *padding* the input image with zeros. In many cases, using a padding of $p_h = k_h - 1$ and $p_w = k_w - 1$ is sufficient, which will result in an output shape matching the input shape. [32, p. 254-255]

Stride

The *stride* refers to the number of rows and columns that the kernel moves at each step. In some scenarios, we may want to downsample the input image for e.g. computational efficiency reasons. This can be achieved by using a stride larger than 1, which results in skipping intermediate locations yielding a smaller output shape. A stride of s_h and s_w will result in an output shape of $\lfloor (n_h - k_h + p_h + s_h)/s_h \rfloor \times \lfloor (n_w - k_w + p_w + s_w)/s_w \rfloor$. [32, p. 256-257]

2.3.2 Pooling

Another core concept of CNNs is *pooling*. Goodfellow et al. [13, p. 339] define pooling as a function which *"replaces the output of the net at a certain location with a summary statistic of the nearby outputs"*. The main purpose of pooling is to make the *"representation approximately invariant to small translations of the input"* [13, p. 342]. A small translation in the input image should not result in a large change in the output of the pooling layer. Deciding if an object (e.g. face) is present in an image should not depend on the location of the object in the image. [13, p. 342]

Commonly used pooling functions are *maximum pooling* and *average pooling*.

Maximum pooling obtains the maximum value of the input in the pooling area, while average pooling obtains the average value of the input in the pooling area. [32, p. 264]

In addition to the local translation invariance property, pooling also reduces the computational burden of the network. This is because pooling region summarizes the values in the region. Spacing these pooling regions k pixels apart reduces the inputs to the next layer by a factor of k. This results in reduced memory consumption and improved statistical efficiency. Pooling also allows to handle inputs of varying sizes, which is a useful property for image data. [13, p. 342]

2.3.3 Architecture

A *'convolutional layer'* consists of a sequence of operations – convolution, nonlinearity and pooling [13, p. 341]. The input image is processed by k different kernels, representing the weights and connections in the CNN. These convolution (2.3.1) kernels produce a *feature map* for each kernel. Finally, the feature maps are pooled (2.3.2) to reduce the dimensionality of the feature maps. [25, p. 122-132]

CNNs are typically composed of stacks of relatively complex convolutional layers [13, p. 341]. An example of a CNN architecture is shown in Figure 2.5. Because pooling is inherently a destructive operation, we tend to stack multiple convolutional layers before pooling in deeper CNNs [25, p. 132]. Convolutional layers are typically followed by fully-connected layers, which are used to map the high-level features to the desired output. Fully-connected layers can compress the flattened feature maps into a vector of desired length. [25, p. 135]

In a traditional fully-connected neural-network, each output unit interacts with all the input units. However, CNNs use *sparse interactions*, i.e. each output unit interacts with only a subset of the input units, which is achieved with a tiny kernel compared to the input size. Fewer connections result in fewer parameters, which

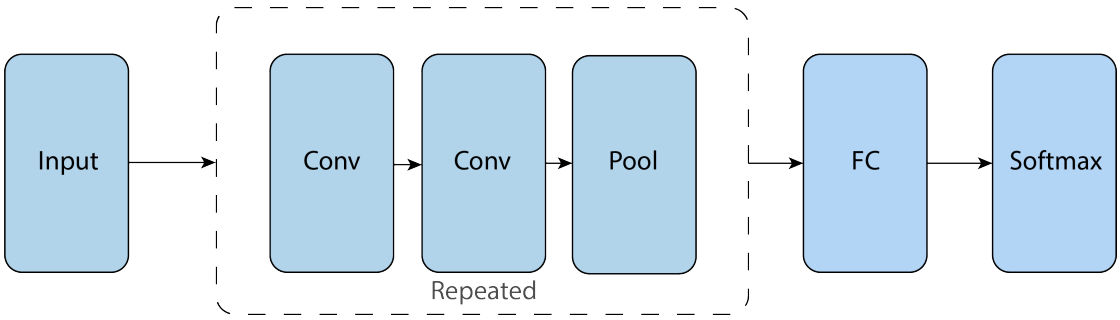

Figure 2.5: An example of a generic CNN architecture, adapted from [25, p. 133].

reduces the memory requirements and improves statistical efficiency. In deeper CNNs, the deeper layers can *'see'* a large area of the input image *indirectly* through the layers, allowing it to efficiently learn complex representations with relatively few parameters. [13, p. 335]

Another key property of CNNs is *parameter sharing*. Traditionally, each weight in a fully-connected neural-network is used only once when computing the output. However, in CNNs, each kernel weight is applied to all positions of the input image (except for the edges). Thus, the same kernel parameter is used for multiple computations, and only a set of parameters is learned. [13, p. 335-338]

3 Object Detection

Vision is a task that is natural for humans, yet difficult for computers and machines. *Computer Vision* (CV) is and has been an active research field, and lately especially important with the rise of DL. As a field, CV is very broad as it includes diverse ways of processing images and widely different application possibilities. [13, p. 452]

In traditional image classification, the task is to classify an image with a category, based on the key or major object in the image. However, in many applications there are multiple objects in an image, requiring the computer to know both the categories and the locations of the objects in the image. This task is called *object detection*. Spatial location of an object is usually represented with a *bounding box*, a rectangle commonly defined by the x and y coordinates of the top-left corner and the bottom-right corner. [32, p. 629-630]

Modern Object Detection

Object detection pipeline consists of three stages, *informative region selection*, *feature extraction*, and *classification* [33]. Applying traditional recognition algorithms in image recognition can be too slow and inaccurate to be practical. Constructing special-purpose detectors, that are able to find likely locations of objects in an image quickly, is a more effective approach for object detection. [42, p. 295] Modern object detectors rely on CNNs, which considerably reduce the amount of computation needed in comparison to traditional recognition algorithms [17]. In the DL era of

object detection, the detectors are roughly categorized into two groups: *two-stage* (Section 3.1) and *one-stage* (Section 3.2) detectors [3]. Modern object detectors use a *backbone* network, e.g. ResNet (Section 3.3.1), for feature extraction.

3.1 Two-Stage Detectors

Two-stage detectors frame the object detection as a two-stage *"coarse-to-fine"* process [3]. The first stage of a two-stage detector is to propose a set of plausible rectangular regions in the image, called region proposals. A classifier is then applied to each region proposal for determination of object presence, and for refinement of the bounding box of the detected object. [42, p. 304].

R-CNN

As deep CNNs proved out to be very effective in learning high-level feature representations of an image [3], in 2014 Gircshick et al. a proposed a two-stage object detector called *R-CNN* (Region-based CNN) [43]. R-CNN extracts region proposals from an image using a *selective search* [44] algorithm, which are then fed into a CNN to extract features [3]. At the end, one linear SVM (Support Vector Machine) is trained for each object category for classification. At the time, R-CNN yielded a substantial 30% relative improvement over the existing models on the PASCAL VOC 2012 object detection challenge. [43]

Fast R-CNN

Fast R-CNN [36], proposed a year later by Girshick, is an improved version of R-CNN [43], which includes innovative improvements yielding a 9× improvement in training time of VGG16 network and a 213× improvement in inference time over R-CNN, while also improving the detection accuracy. The key innovation of Fast R-CNN is the *Region of Interest (RoI) pooling layer*. First, a CNN processes the

entire image to produce a convolutional feature map. Using the RoI pooling layer, the feature map allows the extraction of a fixed-length feature vector for each region proposal. [36]

Max-pooling is applied to each valid RoI to convert the features into a small feature map with $H \times W$ (e.g. 7×7) shape. The feature extraction can be done in a single forward pass of the CNN, allowing the RoIs to share memory and computation burden in both forward and backward passes. [36]

3.1.1 Faster R-CNN

Faster R-CNN [45] is an object detection model that builds on the R-CNN [43] and Fast R-CNN [36] detectors. Fast R-CNN successfully made the detection network faster, but the region proposal step remained as a bottleneck [45]. Faster R-CNN alleviates this bottleneck by introducing a *Region Proposal Network* (RPN), a Fully Convolutional Network (FCN) predicting object bounding boxes and objectness scores at each location simultaneously [45].

Region Proposal Network

The RPN receives an image as input and outputs a collection of region proposals with objectness scores. Generation of the region proposals is done by applying a smaller network on the convolutional feature map. This small network receives a $n \times n$ spatial window of the convolutional feature map (originally $n = 3$). Projection to a lower-dimensional feature map is performed, and two fully-connected layers are applied to the projected feature map. These fully-connected layers are box-regression and box-classification layers. All spatial locations share these fully-connected layers, making the RPN efficient. [45]

For achieving translational invariance and for tackling the issue of different scales of objects, the RPN uses *multi-scale anchors as regression references*. For each

sliding window position, maximum k proposals are generated. Reference boxes, called *anchors*, of various aspect ratios and scales are defined at each sliding window position. The region proposals of the RPN are parameterized relative to these k anchors. Originally, Ren et al. used $k = 9$ anchors at each sliding position, consisting of 3 scales and 3 aspect ratios. The anchor-based approach allows sharing of computation between the different scales and aspect ratios, as the convolutional feature map is computed only once for the entire image. [45]

The loss function of the RPN is defined as a sum of classification and regression loss [45]:

$$L(\{p_i\}, \{t_i\}) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*) \tag{3.1}$$

where L_{cls} is the binary cross-entropy loss (object or not), L_{reg} is the smooth L_1 loss for bounding box regression. Terms p_i and p_i^* are the predicted probability and ground-truth label of anchor i and terms t_i and t_i^* are the predicted and ground-truth bounding box coordinates of anchor i. Additionally, the loss function includes normalization terms N_{cls} and N_{reg}, and a balancing term λ. The RPN can be trained with stochastic gradient descent (SGD), and Ren et al. used momentum optimization with a momentum of 0.9. [45]

Architecture and Training

Faster R-CNN consists of two ‘modules‘, the first being the RPN and the second being the Fast R-CNN [36], which utilizes the region proposals generated by the RPN. The architecture of Faster R-CNN is a unified network such that the convolutional layers are shared between the RPN and the Fast R-CNN detector. [45]

The overall loss function of Faster R-CNN is the sum of the RPN loss and the Fast R-CNN loss. The Fast-RCNN uses a multi-task loss function [36]:

$$L(p, u, t^u, v) = L_{cls}(p, u) + \lambda[u \geq 1]L_{reg}(t^u, v) \tag{3.2}$$

where L_{cls} is the categorical cross-entropy loss and L_{reg} is the smooth L_1 loss. Terms p and u are the predicted probability and ground-truth label of a RoI, and terms t^u and v are the predicted and ground-truth bounding box coordinates of a RoI. Additionally, the loss function includes a balancing term λ and an indicator function $[u \geq 1]$ for handling the catch-all background class (i.e. $u = 0$).

Joint training of the RPN and the Fast R-CNN detector requires a specific technique the authors used, called *alternating training*. In alternating training, the RPN is trained first and then the Fast R-CNN detector is trained with the proposals of the RPN. The RPN is then initialized with the network tuned by Fast R-CNN detector and the alternating training continues. [45]

3.2 One-Stage Detectors

Two-stage detectors can achieve high precision, but the complexity of the two-stage pipeline results in high requirements for computational resources. In contrast, one-stage detectors treat object detection as a *"complete in one step"* problem. Generally, one-stage detectors are faster than two-state detectors, which makes them more suitable for applications with real-time requirements. However, their precision is usually worse especially when detecting smaller objects. [3]

3.2.1 Single-Shot Multibox Detector

The Single-Shot Multibox Detector (SSD) [46], proposed by Liu et al. in 2016, is a one-stage object detector that achieves high average precision with high detection speed. SSD is relatively simple compared to two-stage approaches, because no explicit region proposal step is used. The idea of SSD is to use a collection of

predefined bounding boxes to predict category scores and bounding box offsets. [46]

Architecture

The SSD architecture builds on a generic backbone network (originally VGG16 [47]) while including additional convolutional layers. The convolutional layers progressively decrease in size, due to pooling, yielding a set of feature maps of different sizes, allowing detection of objects of different scales. A small convolutional kernel size of 3×3 is the basic element for predicting category scores and bounding box offsets. The bounding box offsets are predicted relative to the default bounding boxes. [46]

Default Boxes

A collection of default bounding boxes, often called *default boxes*, are defined and anchrored relative to each feature map location. For each box out of k default boxes, the SSD predicts offsets (4) to the original default box shape and c class scores (where c is object category count). The authors used $k = 6$ default boxes with different aspect ratios $(1, 2, 3, \frac{1}{2}, \frac{1}{3})$ and scales. The idea of default boxes is similar to the anchors used in Faster R-CNN. The key difference is that the predefined boxes are applied to feature maps of different resolutions. [46]

Training

Training of SSD requires matching of ground truth information to the chosen default boxes. To determine the assignment, the authors used a matching strategy, where the predefined boxes and ground truth boxes are matches based on the *jaccard* overlap, i.e. the *Intersection-over-Union* (see Section 3.4.1). [46]

The loss function of SSD is a weighted sum [46]:

$$L(x, c, l, g) = \frac{1}{N}(L_{conf}(x, c) + \alpha L_{loc}(x, l, g)) \tag{3.3}$$

where L_{conf} is the confidence loss (softmax over class confidences), L_{loc} is the localization loss (smooth L_1 loss), x is the input data, c is the class predictions, l is the predicted bounding box, g is the ground truth bounding box, and α is a balancing parameter.

3.3 Feature Extraction

3.3.1 ResNet

There exists many different deep CNN based feature extraction (commonly known as *backbone*) architectures for object detection, such as ResNet [48], VGG [47] and others. In this thesis, the ResNet architecture is used as the backbone for the object detection models.

While deeper NNs have more expressive power, they are more difficult to train because of *vanishing* or *exploding* gradients. The more layers the network has, the more multiplications the gradient has to go through, which can result in the gradient vanishing, hurting the convergence.

Stacking many layers on top of each other has introduced a *degradation* problem, where the NN accuracy is saturated and degrades rapidly. He et al. [48] propose a solution to this problem with the *Residual Network* (ResNet) architecture. The idea of ResNet is to add *skip connections* between layers, which allows the network to learn *residual* mappings instead of the direct mappings. [48]

A *residual block* is a fundamental core of ResNet. It consists of a series of weight (e.g. convolutional) layers and a *skip connection* that adds the input of the block to the output of the block, bypassing the weight layers. Let x be the input and $f(x)$ be the direct mapping to learn. In a residual block, the weight layers learn the residual

$g(x) = f(x) - x$ instead of the desired $f(x)$. Thus, the output is $f(x) = g(x) + x$, where x is from the *skip connection*. [32, p. 312]

The original ResNet architecture was based on plain 34-layer CNN, to which the skip connections were added. The ResNet34 architecture had better performance than the baseline plain 34 layer CNN, and the degradation problem was addressed well. [48] More commonly known variations of the ResNet architecture are ResNet50, ResNet101 and ResNet152, which are based on 50, 101 and 152 layers respectively.

3.3.2 Feature Pyramid Network

Detection of objects at different scales is a common problem not only in maritime vessel detection, but in object detection in general. The *Feature Pyramid Network* (FPN) [49] is a feature extraction architecture that aims to allow the network to better detect objects at different scales. FPN is based on an idea of a *pyramid* of feature maps of different scales. The FPN has two *pathways* – a bottom-up pathway and a top-down pathway. Each of the stages of the bottom-up pathway is connected to the top-down pathway via lateral connections, allowing the top-down pathway to use the feature maps of different scales from the bottom-up pathway. [49]

The bottom-up pathway is the standard feature extraction backbone, such as ResNet, producing feature maps of different scales. For ResNets, the *stages* (or levels) of the pyramid are produced by the last residual block of each stage. [49]

The top-down pathway essentially hallucinates higher-resolution feature maps from the coarser feature maps of the bottom-up pathway. The top-down pathway produces semantically strong, but spatially coarse feature maps, later refined with the information from the lateral connections. [49]

In the context of this thesis, the original SSD inherently uses a pyramidal feature hierarchy, as it uses feature maps of different scales for detection. The difference is that SSD builds the feature pyramid high-up in the network, i.e. not reusing the

higher-resolution feature maps of the hierarchy. The FPN is a more general and flexible architecture, showing to improve the performance, especially in small object detection. [49]

3.4 Evaluation Metrics

3.4.1 Intersection over Union

In order to assess the performance of an object detection model, a metric for *"correct detection"* is needed. Intersection over Union (IoU) measures the overlap between two bounding boxes. Mathematically, IoU is defined as the ratio of the intersection area and the union area of two bounding boxes [50]:

$$\text{IoU} = \frac{\text{Area of Intersection}}{\text{Area of Union}} = \frac{\text{area}(B_p \cap B_{gt})}{\text{area}(B_p \cup B_{gt})} \tag{3.4}$$

where B_p is the predicted bounding box and B_{gt} is the ground truth bounding box. A visual example of the IoU is shown in Figure 3.1.

IoU allows to establish definitions for *"correct detection"* and *"incorrect detection"*. Given a threshold t and an IoU value of B_p and B_{gt}, a detection is considered correct if the IoU is greater than or equal to the threshold t [50]:

$$\text{IoU} \geq t \iff \text{Correct Detection} \tag{3.5}$$

3.4.2 Precision and Recall

Confusion Matrix (CM), is used to describe the outputs of a classification model [51]. Viewing object detection as a binary classification problem between *"correct"* and *"incorrect"* detections, we can extrapolate from the CM the following information [50]:

Figure 3.1: IoU (0.80) of a vessel detection, image from SeaShips and overlays by the author – ground truth bounding box in green, predicted bounding box in red.

- True Positive (TP): A correct detection of B_{gt}
- False Positive (FP): An incorrect detection of B_{gt}
- False Negative (FN): No detection of B_{gt}
- True Negative (TN): Not applicable in object detection (infinite number of B that should not be detected)

CM and the definitions above are the basis of many different metrics [51]. In object detection, the assessment of performance is largely based on the *precision* and *recall* metrics [50].

Precision is defined mathematically as the ratio of the correct detections (TP) to the total number of detections (TP + FP) [50]:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{3.6}$$

Recall is defined as the ratio of the correct detections (TP) to the total number of ground truth objects (TP + FN) [50]:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3.7}$$

Intuitively, precision measures the ability to identify *only* relevant objects, while recall measures the ability to find *all* relevant objects [50].

Precision-Recall Curve

Precision and recall usually have an inverse relationship, i.e. increasing one decreases the other. Generally, if FP is low, precision will be higher, but recall is lower. Likewise, if FN is low, recall will be higher, but precision is lower. An ideal object detector would have both high precision and high recall (FN = 0 and FP = 0). Precision-recall curve is a graph that shows the relationship between precision and recall for different confidence thresholds t. The area under the precision-recall curve (AUC) is a metric summarizing the overall performance of the model, where higher AUC indicates better performance. [50]

Average Precision

In practical applications of object detection, the precision-recall curve has a zigzag-like shape, which makes it difficult to accurately measure the AUC [50]. The AUC can be summarized with the *average precision* (AP) metric, which is computed using *interpolation*. For example, in 11-point interpolation the precision is measured at

11 equally spaced recall levels $R \in \{0, 0.1, ..., 0.9.1\}$, yielding the AP [50]:

$$\mathrm{AP}_{11} = \frac{1}{11} \sum_{R \in \{0,0.1,...,0.9.1\}} P_{\mathrm{interp}}(R) \tag{3.8}$$

where $P_{\mathrm{interp}}(R)$ is the maximum precision of all recall levels greater than R.

Mean Average Precision

Mean Average Precision (mAP) is an extension of the AP metric, which summarizes the AP over multiple object categories. Let C be the set of object classes, then the mAP is defined as [50]:

$$\mathrm{mAP} = \frac{1}{|C|} \sum_{c \in C} \mathrm{AP}_c \tag{3.9}$$

3.4.3 COCO Detection Evaluation Metrics

This thesis uses the COCO Detection Evaluation [52] metrics for evaluating the performance of the object detection models. In total, 12 different metrics in 4 groups are used, as shown in Table 3.1.

Metric Group	Metric	Description
Average Precision (AP)	AP	AP at IoU = 0.50:0.05:0.95
	$\mathrm{AP}^{\mathrm{IoU=.75}}$	AP at IoU = 0.50
	$\mathrm{AP}^{\mathrm{IoU=.50}}$	AP at IoU = 0.75
AP Across Scales	$\mathrm{AP}^{\mathrm{small}}$	AP small objects (area $< 32^2$)
	$\mathrm{AP}^{\mathrm{medium}}$	AP medium objects ($32^2 <$ area $< 96^2$)
	$\mathrm{AP}^{\mathrm{large}}$	AP large objects (area $> 96^2$)
Average Recall (AR)	$\mathrm{AR}^{\mathrm{max=1}}$	AR given 1 detection per image
	$\mathrm{AR}^{\mathrm{max=10}}$	AR given 10 detections per image
	$\mathrm{AR}^{\mathrm{max=100}}$	AR given 100 detections per image
AR Across Scales	$\mathrm{AR}^{\mathrm{small}}$	AR small objects (area $< 32^2$)
	$\mathrm{AR}^{\mathrm{medium}}$	AR medium objects ($32^2 <$ area $< 96^2$)
	$\mathrm{AR}^{\mathrm{large}}$	AR large objects (area $> 96^2$)

Table 3.1: The evaluation metrics of the COCO detection challenge [52]. The authors use the terms mAP and AP (as well as mAR and AR) interchangeably.

4 Experiment

In order to understand and measure the capabilities of Transfer Learning in the context of vessel detection from waterborne images, this thesis presents an experiment in which two pre-trained object detectors, SSD and Faster R-CNN, are trained and evaluated on two datasets, ABOships and SeaShips. The experiment outline is presented in Section 4.1. The datasets are described in Section 4.2, the object detectors in Section 4.3, the training process in Section 4.4 and the evaluation process in Section 4.5. The experiment is conducted on a single consumer-grade computer with the specifications presented in Section 4.6.

4.1 Outline

The main outline of the experiment is to fine-tune two pre-trained object detectors, SSD and Faster R-CNN, on two maritime datasets, 'ABOships' and 'SeaShips'. Different ResNet feature extractors are used to study their effect on the transfer learning performance. The research questions of the thesis have been presented in Section 1.1.2, and for convenience are repeated here:

Research Questions

1. How does the vessel detection performance differ between the two object detector architectures, one-stage SSD FPN (3.2.1) and two-stage Faster R-CNN (3.1.1)?

2. How does the performance of the object detectors differ when fine-tuned on the maritime datasets, ABOships (1.3.2 & 4.2.2) and SeaShips (1.3.3 & 4.2.3)?

3. How do different ResNet (3.3.1) feature extractors sizes affect the performance of SSD FPN and Faster R-CNN object detectors?

4.2 Datasets

The experiment is conducted on two maritime datasets, ABOships [18] and SeaShips [21]. The base models have been trained on the generic COCO [19] dataset, introduced in Section 1.3. In order to evaluate the unbiased performance of the vessel detectors, the datasets are split into three subsets: training, validation and test. The training set is used to train the object detectors, the validation set is used to evaluate the performance of the object detectors during training and the test set is used to produce an unbiased estimate of the performance of each vessel detector.

4.2.1 COCO

For the purposes of this thesis, the COCO dataset contains 3 146 images of vessels and has been discussed in Section 1.3.1. The drawbacks of COCO in the context of vessel detection are the small amount of images containing vessels and the lack of information about the categories of vessels. Examples of the ‘boat‘ category are shown in Figure 1.2. Yet, it is suitable for pre-training the object detectors, as it contains a large amount of images and object instances. All models (4.3) used in the thesis have been pre-trained on the COCO dataset 2017 edition and are publicly available in the TensorFlow 2 Detection Model Zoo [53].

4.2.2 ABOships

Each annotated maritime object (41968 in total) in the ABOships dataset belongs to one of the 11 classes: ‘boat‘, ‘cargoship‘, ‘cruiseship‘, ‘ferry‘, ‘militaryship‘, ‘miscboat‘, ‘miscellaneous‘, ‘motorboat‘, ‘passengership‘, ‘sailboat‘ and ‘seamark‘. For the purposes of this thesis, the categories ‘miscellaneous‘ and ‘seamark‘ are excluded from the dataset, as they are not vessels of any kind or have not been identified as such. After the exclusions, the dataset contains 7 992 images of vessels, 34 100 instances in total. The category distribution of the dataset is shown in Figure 4.1 and an example image in Figure 1.3.

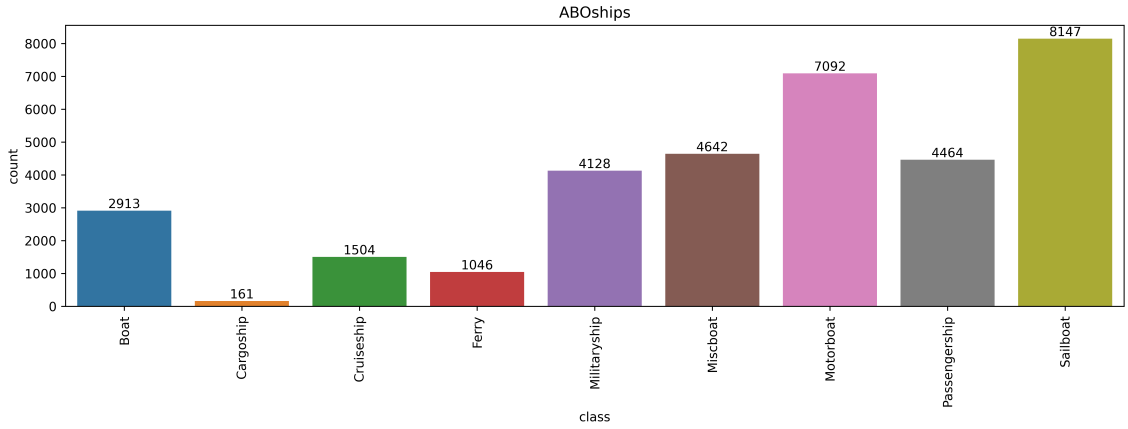

Figure 4.1: The category distribution of the ABOships dataset, where the categories ‘miscellaneous‘ and ‘seamark‘ have been excluded.

The extraction process described in the original article yielded a dataset with images of vessels from different angles and distances, as well as in different weather and lighting conditions. Variety is a desirable characteristic for the purposes of this thesis, as the object detectors are expected to perform well in different conditions. One consideration is the dependency between the video frames. However, the extraction interval being 15 seconds, the dependency between the frames should be minimal and not expected to have a significant effect on the results.

Data Split

The ABOships dataset is not distributed with a predefined data split, so for the purposes of the experiment the dataset needs to be split into three subsets: training, validation and test. The dataset is split into three subsets with a ratio of 70% (5 333 images, 22 907 vessels) for training, 15% (1 329 images, 5 594 vessels) for validation and 15% (1 330 images, 5 599 vessels) for testing. In order to preserve the distribution of the classes in the subsets, special care is taken when splitting the dataset. First the labels are grouped by image filename, then the groups are split into the three subsets using stratified random sampling. Grouping by filename ensures that an image cannot be split into multiple subsets, which would cause data leakage, invalidating the results. Stratified sampling ensures that the distribution of the classes is preserved also in the subsets. The class distributions of the subsets are shown in Figure 4.2 and of the original dataset in Figure 4.1.

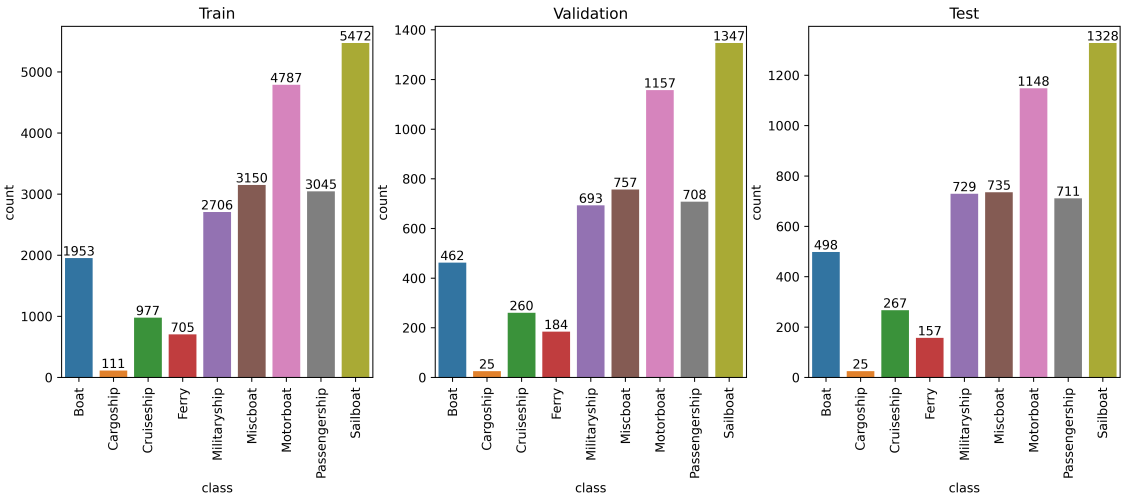

Figure 4.2: The class distributions of the subsets of the ABOships dataset using stratified random sampling.

4.2.3 SeaShips

The publicly available subset [22] of the SeaShips dataset is the other maritime dataset used in the experiment. The subset contains 7 000 images of vessels from the original dataset. The vessels are divided into six categories: ‘ore carrier‘, ‘bulk cargo carrier‘, ‘general cargo ship‘, ‘container ship‘, ‘fishing boat‘ and ‘passenger ship‘. The category distribution of the dataset is shown in Figure 4.3 and an example image in Figure 1.4.

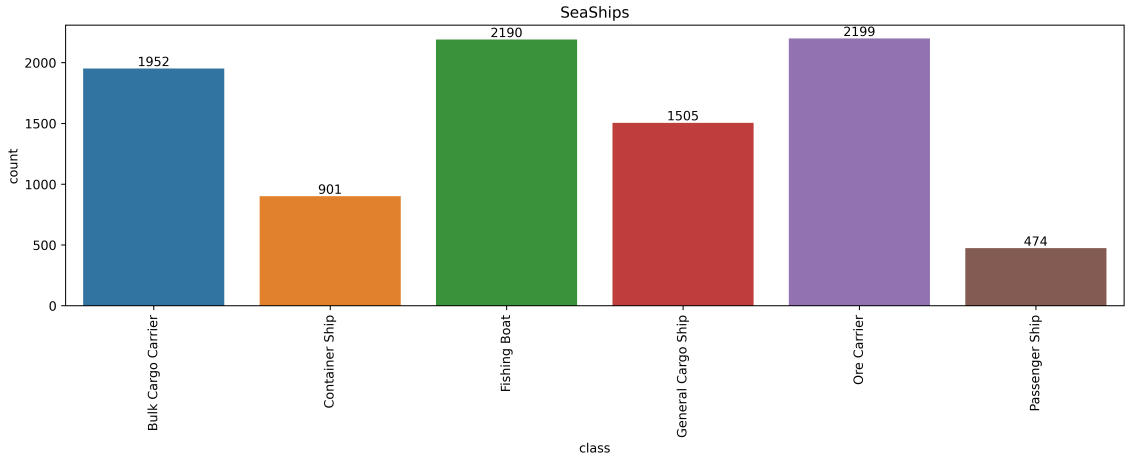

Figure 4.3: The category distribution of the SeaShips (7000) dataset.

Data Split

While the publicly available subset of the SeaShips dataset is distributed with a predefined data split, an explicit data split is performed for the purposes of the experiment. The predefined data split consists of 25% (1 750 images) for training, 25% (1 750 images) for validation and 50% (3 500 images) for testing. In order to achieve comparable results with the ABOships dataset, the dataset is split into three subsets with a ratio of 70% (4 668 images, 6 134 vessels) for training, 15% (1 166 images, 1 550 vessels) for validation and 15% (1 166 images, 1 537 vessels) for testing. The split procedure is the same as with the ABOships dataset, grouping by filename and using stratified random sampling to ensure that the distribution of the

classes is preserved in the subsets. The class distributions of the subsets are shown in Figure 4.4 and of the original dataset in Figure 4.3.

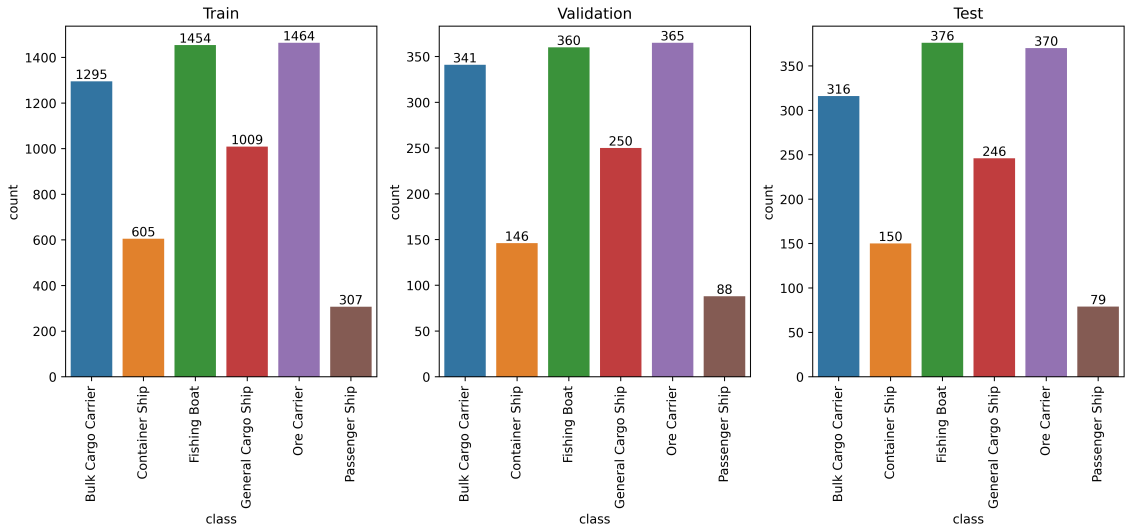

Figure 4.4: The class distributions of the subsets of the SeaShips dataset using stratified random sampling.

4.3 Models

The experiment is conducted using two pre-trained object detectors, SSD (Section 3.2.1) and Faster R-CNN (Section 3.1.1). The SSD models use FPN (Section 3.3.2) with the backbone feature extractors. These pre-trained object detectors are available in the TensorFlow 2 Detection Model Zoo [53] with various feature extractors, of which ResNet50, ResNet101 and ResNet152 are used in the experiment. The ResNet feature extractors are named according to the number of layers in the network, e.g. ResNet50 has 50 layers. All object detectors have been pre-trained on the COCO dataset and are fine-tuned separately on the ABOships and SeaShips datasets. The configurations used in the experiment are presented in Table 4.1.

The pre-trained models are distributed with a configuration file, which contains the definition of the model architecture and the hyperparameters used in training. The configuration files are kept largely unchanged, with the exception of slight

Detector	Feature Extractor	Input Size
SSD	ResNet50 FPN	640x640
SSD	ResNet101 FPN	640x640
SSD	ResNet152 FPN	640x640
Faster R-CNN	ResNet50	640x640
Faster R-CNN	ResNet101	640x640
Faster R-CNN	ResNet152	640x640

Table 4.1: The configurations of the pre-trained object detectors used in the experiment.

modifications to the hyperparameters. The training process is described in more detail in Section 4.4.

4.4 Training

Preparation

Before training, all datasets are converted into the TensorFlow's *TFRecord* [54] format. The images and annotations of the datasets are split into multiple TFRecord files, i.e. sharded, for performance reasons.

Configuration

For all models, the minibatch size is reduced to 4 from the pre-configured 64, in order to not exceed the memory capacity of the GPU. A drastic change in batch size should be taken into account when configuring the learning rate, according to Krizhevsky [55, p. 5]. As a heuristic rule, Kirzhevsky suggests to multiply the learning rate by k when multiplying the batch size by k. In this case, the learning rate is multiplied by $\frac{4}{64} = \frac{1}{16} = 0.0625$, as the batch size is reduced from 64 to 4. Both the warmup phase and the decay phase of the learning rate schedule are adjusted accordingly. Experimentation in the context of this thesis showed that the learning rate change was necessary in order to achieve stable training.

All models in this thesis are optimized using the Momentum (SGD) optimizer

with a momentum value of 0.9, which is the pre-configured default value in the model configuration files. The default cosine decay learning rate schedule, which consists of a warmup phase and a decay phase, is used for all models. The warmup phase is used to gradually increase the learning rate to the base learning rate, and lasts for 2000 steps. The decay phase is used to gradually decrease the learning rate and lasts for 23 000 steps, making the total training of a model to last for 25 000 steps.

Data augmentation is disabled for all models, as the aim of the experiment is to study the raw performance in the context of vessel detection. The chosen object detectors are fine-tuned using the training scripts of the TensorFlow 2 Object Detection API [56] and the training process is monitored using TensorBoard [57].

4.5 Evaluation

The evaluation is performed using the COCO Detection Evaluation metrics (see Section 3.4.3). This is achieved by modifying the configuration file of the object detectors to use said metrics. The evaluation is executed using the evaluation scripts of the TensorFlow 2 Object Detection API, similarly to the training process.

In the experiment, continuous evaluation with the validation set is used to monitor the performance of the vessel detectors during training. The validation is performed at every 1 000 steps.

The out-of-sample performance of the vessel detectors is evaluated using the test set. The results of the out-of-sample evaluation are presented in Chapter 5. Cross-validation is not used in the experiment. This is because of the lack of the computational resources required to perform cross-validation, as well as the test set being large enough to produce a reliable point estimate of the performance of the vessel detectors.

4.6 Environment

The experiment is conducted on a single consumer-grade computer with the following specifications:

CPU	AMD Ryzen 7 7800X3D
GPU	NVIDIA GeForce RTX 4080 16 GB GDDR6X
RAM	64 GB DDR5

The following operating system, programming language and libraries are used:

	Version
Ubuntu (Windows 11 Pro WSL2)	23.04
Python	3.10.13
TensorFlow & Object Detection API [56]	2.6.0
CUDA Toolkit	11.8
cuDNN	8.9.5.29
TensorRT	8.6.1

5 Experiment Results

Chapter 5 presents the results of the experiment described in Chapter 4, attempts to answer the research questions posed in Section 1.1.2 and presents the challenges in maritime vessel detection. Section 5.4 reflects on the results, compares the results to other work, and presents discussion about potential future work. Figures 5.1 and 5.2 show examples of the vessel detections from the ABOships and SeaShips test sets, respectively.

5.1 Results

Section 5.1 presents the numerical results of the experiment. Both precision and recall metrics are shown for each of the detectors and datasets. Out of sample performance is measured using a test set that was not used during training, as described in Section 4.2.

Overall, the experiment's results are mostly inline what was expected from the experiment setting, albeit some of the results are surprising.

5.1.1 Precision

COCO precision metrics (see Section 3.4.3) for ABOships dataset are presented in Table 5.1 and for SeaShips in Table 5.2.

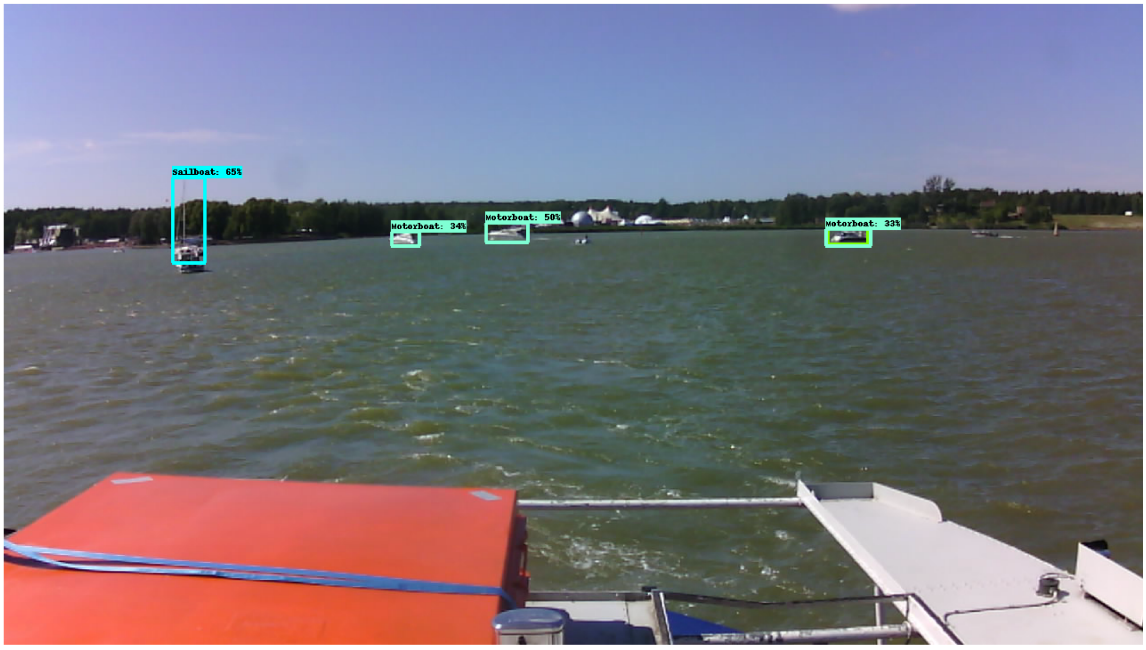

Figure 5.1: Vessel detection example from ABOships test set using SSD FPN with ResNet 101 backbone.

ABOships

ABOships precision metrics are relatively low, which is expected since the dataset is challenging in terms of small vessel sizes and occlusions. All detectors perform similarly, although SSD tends to perform better than Faster RCNN, regardless of the ResNet backbone size. This is likely because the SSD with FPN is designed to extract multiple feature maps at different scales at different levels of the network, whereas Faster RCNN only extracts features at the backbone network. In the case of Faster RCNN, there is some evidence that the larger ResNet backbones increase

Figure 5.2: Vessel detection example from SeaShips test set using SSD FPN with ResNet 101 backbone.

the performance of the detector. For SSD, such evidence is not clearly present.

SeaShips

In SeaShips, the precision metrics are much higher than in ABOships, which indicates that the dataset is easier for the detectors. The results are consistent across the detectors, with SSD performing better than Faster RCNN. The AP^{small} metric shows larger variance than the equivalent metric in ABOships.

Detector	**AP**	**AP**$^{\text{IoU=.75}}$	**AP**$^{\text{IoU=.50}}$	**AP**$^{\text{large}}$	**AP**$^{\text{medium}}$	**AP**$^{\text{small}}$
Faster RCNN 50	0.2470	0.1807	0.5510	0.4000	0.2496	0.0980
Faster RCNN 101	0.2537	0.1951	0.5615	0.3944	0.2675	0.0991
Faster RCNN 152	0.2612	0.2051	0.5647	0.4184	0.2732	**0.1142**
SSD 50 FPN	0.2654	0.2155	0.5720	0.4131	0.2841	0.1003
SSD 101 FPN	**0.2772**	**0.2367**	0.5750	**0.4198**	**0.2976**	0.0995
SSD 152 FPN	0.2725	0.2243	**0.5764**	0.4073	0.2898	0.1003

Table 5.1: COCO precision metrics of each of the detectors on the ABOships test set.

Detector	**AP**	**AP**$^{\text{IoU=.75}}$	**AP**$^{\text{IoU=.50}}$	**AP**$^{\text{large}}$	**AP**$^{\text{medium}}$	**AP**$^{\text{small}}$
Faster RCNN 50	0.7517	0.8956	**0.9851**	0.7685	0.4861	0.1837
Faster RCNN 101	0.7620	0.8944	0.9826	0.7786	0.5150	0.1584
Faster RCNN 152	0.7630	0.8949	0.9834	0.7775	0.6007	0.0673
SSD 50 FPN	0.7688	**0.9063**	0.9694	0.7798	**0.6702**	0.1005
SSD 101 FPN	**0.7740**	0.9020	0.9729	**0.7850**	0.6591	**0.2337**
SSD 152 FPN	0.7697	0.9032	0.9720	0.7809	0.5794	0.0950

Table 5.2: COCO precision metrics of each of the detectors on the SeaShips test set.

5.1.2 Recall

COCO recall metrics (see Section 3.4.3) for ABOships dataset are presented in Table 5.3 and for SeaShips in Table 5.4.

ABOships

Detector	**AR**$^{\text{max=1}}$	**AR**$^{\text{max=10}}$	**AR**$^{\text{max=100}}$	**AR**$^{\text{large}}$	**AR**$^{\text{medium}}$	**AR**$^{\text{small}}$
Faster RCNN 50	0.2823	0.3972	0.4193	0.5874	0.4332	0.2624
Faster RCNN 101	0.2837	0.3996	0.4218	0.5990	0.4440	0.2708
Faster RCNN 152	0.2963	0.4127	0.4332	**0.6255**	0.4527	0.2935
SSD 50 FPN	0.2956	0.4386	0.4585	0.5853	0.4769	0.3133
SSD 101 FPN	0.3000	0.4458	0.4635	0.5727	0.4863	**0.3305**
SSD 152 FPN	**0.3013**	**0.4484**	**0.4679**	0.5892	**0.4931**	0.3058

Table 5.3: COCO recall metrics of each of the detectors on the ABOships test set.

Recall results for ABOships further confirm the difficulty of the dataset, with all detectors having a modest recall. Moreover, the recall results validate the con-

clusions from the precision results, with SSD performing better than Faster RCNN. Similar effect of the ResNet backbone size is observed for Faster RCNN, where detectors with larger backbones tend to perform better. For SSD, the effect is not as clear.

SeaShips

Detector	$\mathbf{AR}^{\text{max=1}}$	$\mathbf{AR}^{\text{max=10}}$	$\mathbf{AR}^{\text{max=100}}$	$\mathbf{AR}^{\text{large}}$	$\mathbf{AR}^{\text{medium}}$	$\mathbf{AR}^{\text{small}}$
Faster RCNN 50	0.7363	0.8064	0.8079	0.8245	0.5789	0.2333
Faster RCNN 101	0.7473	0.8161	0.8172	0.8319	0.5959	0.2333
Faster RCNN 152	0.7518	0.8187	0.8196	0.8331	0.6568	0.0666
SSD 50 FPN	0.7536	0.8268	0.8280	0.8376	**0.7186**	0.2667
SSD 101 FPN	**0.7580**	**0.8298**	**0.8308**	**0.8417**	0.6997	0.3333
SSD 152 FPN	0.7530	0.8261	0.8272	0.8368	0.7151	**0.3667**

Table 5.4: COCO recall metrics of each of the detectors on the SeaShips test set.

Recall results for SeaShips across the detectors are high. Faster RCNN tends to benefit from a larger ResNet backbone. SSD has a higher recall than Faster RCNN. The AR^{small} metric is very low for the Faster RCNN with ResNet 152 backbone, indicating issues with small vessel detection.

5.2 Research Questions

Section 5.2 attempts to answer the research questions posed in Section 4.1. These answers are based on the numerical results presented in Section 5.1.

5.2.1 Detector Architecture

How does the vessel detection performance differ between the two object detector architectures, one-stage SSD FPN and two-stage Faster R-CNN?

Overall, the results from both datasets indicate that SSD performs better than Faster RCNN. This is surprising, as Faster RCNN is generally considered to be a more accurate detector than SSD, while SSD provides faster inference times. The difference in performance is not large, but it is consistent across the datasets and the ResNet backbones. Both detector architectures are suitable for vessel detection.

5.2.2 Datasets

How does the performance of the object detectors differ when fine-tuned on the maritime datasets, ABOships and SeaShips?

Both datasets are challenging for all of the detectors used in the experiment. ABOships is a more challenging dataset with many small vessels and occlusions, while SeaShips remains easier with larger vessels. Notably, the setting of the datasets is very different, one being images mostly from port areas with large commercial vessels (SeaShips), and the other one being images from the Finnish Archipelago with smaller vessels (ABOships).

For building a robust and general vessel detection system, a combination of both datasets would be beneficial. This way the detector would be applicable to a wider range of maritime scenarios, provided that the detector is able to generalize well.

Both datasets include images with varying environmental conditions, such as low-light conditions, which is important for building a robust vessel detection system. For proper night-time vessel detection, a different sensor, such as thermal camera is needed. For challenges in vessel detection using these datasets, see Section 5.3.

5.2.3 ResNet

How do different ResNet (3.3.1) feature extractors sizes affect the performance of SSD FPN and Faster R-CNN object detectors?

The numerical results in Section 5.1 indicate that the ResNet backbone size has an effect on the performance of the detectors, however it is not consistent across the detectors. With Faster RCNN, the larger ResNet backbones perform better. Both precision and recall metrics show increased numerical values with larger ResNet backbones. For example, in ABOships using the Faster RCNN detector architecture, the AP metric increases from 0.2470 to 0.2612 when the ResNet backbone is changed from 50 to 152. Similarly in SeaShips, the AP metric increases from 0.7517 to 0.7630 when the ResNet backbone is changed from 50 to 152.

The effect of the ResNet backbone size with the SSD architecture varies, showing no clear evidence on improvement in precision. The recall metrics show similar results, where the effect of the ResNet backbone size is not as clear as with Faster RCNN.

Altogether, the numerical results indicate that a larger ResNet backbone is beneficial for the detectors. The feature extraction in the Faster RCNN architecture is performed only at the backbone network (cf. SSD with FPN), which is likely the reason for the effect of the ResNet backbone size. However, since SSD does not show as strong evidence of the effect of the ResNet backbone size, simply using a larger ResNet backbone is not a guaranteed way to improve the performance of the detector. A larger ResNet backbone increases the computational cost of the detector, which means both training and inference are more computationally expensive.

5.3 Challenges

5.3.1 Vessel Size and Distance

In vessel detection, the size and distance of the vessels play an important role in detections. A large vessel (such as a cargo ship) can appear to be of similar size as a small vessel (such as a fishing boat) in the image, as seen in Figure 5.3. The

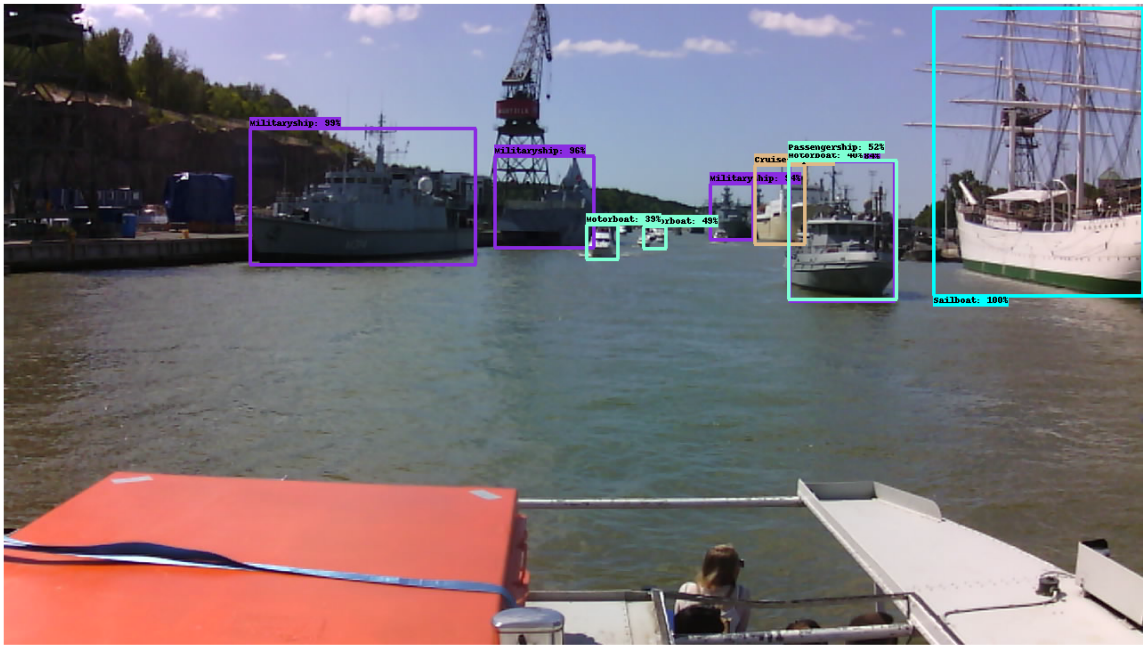

Figure 5.3: An example from ABOships test set using SSD FPN with ResNet 101 backbone. Vessels of different real world sizes appear to be of similar size in the image.

detection of the second military ship from the left (score 0.96) is almost the same size as the passenger boat at the foreground (score 0.52), even though the military ship is much larger in reality. At the open sea, where distances are large, the issue is much more problematic than in port areas. Thus, future research should focus on improving the detection of vessels at larger distances, i.e. improving the detection performance of perceived *'small'* instances.

5.3.2 Occlusions

Figure 5.4: An occlusion example from SeaShips test set using SSD FPN with ResNet 101 backbone. An incorrect and inaccurate detection of 'ore carrier' as 'bulk cargo carrier' due to the occluding 'fishing boat'.

Occlusions are a challenge not only in vessel detection, but in object detection in general. In maritime vessel detection, especially in port areas where the traffic is dense, the vessels are often occluded by other vessels. In the experiment of the thesis, occlusions were a challenge, as seen in Figure 5.4. The 'ore carrier' is occluded by the 'fishing boat', which causes the detector to incorrectly detect the 'ore carrier' as a 'bulk cargo carrier' with an incorrect bounding box.

5.3.3 Environmental Conditions

Figure 5.5: A late evening image from SeaShips test set, with detection score threshold of 0.3. Left: Faster RCNN ResNet 152 with a detection (score $\approx$ 1.0). Right: SSD FPN ResNet 152 without a detection (score $\approx$ 0.2).

Environmental conditions proved out to have a significant impact on the performance of the detectors. Especially low-light conditions, such as late evening and night-time, were problematic in terms of consistent detections. This is especially evident in Figure 5.5. Faster RCNN with ResNet 152 backbone is able to detect the 'ore carrier' with high confidence, while SSD FPN with ResNet 152 backbone is not. This one example already shows the issue with low-light conditions in vessel detection. The 'ore carrier' occupies a large portion of the image, and is clearly visible to the human eye, but is problematic for the detectors.

Low-light vessel detection performance could be improved by using a different sensor, such as thermal camera and sensor fusion. Farahnakian and Heikkonen [7] have explored different sensor fusion architectures (RGB + IR) for vessel detection especially in low-light conditions.

5.4 Discussion and Future Work

The results of this thesis are inline with the baseline results of both ABOships and SeaShips datasets. The research findings of this thesis provide additional value in understanding the performance of detectors and different size ResNet backbones in maritime vessel detection.

ABOships Baseline Comparison

The authors of ABOships [18] have presented baseline detection results with using various detector architectures and backbones pre-trained on the COCO dataset. The results of this thesis are in line with the baseline results, although it is unclear what input size the baseline results refer to (640×640 in this thesis). Notably, the baseline results use a modified definition of the small vessel class ($16^2 <$ area $< 32^2$) versus the COCO definition of small objects (area $< 32^2$). In other words, the authors have excluded the vessels with area smaller than 16^2 pixels, whereas this thesis uses all vessels in the dataset. This difference makes the AP^{small} and AP metrics incomparable between the baseline and this thesis, yet AP^{medium} and AP^{large} remain comparable. Comparison of results of this thesis and the baseline results, where applicable, are shown in Table 5.5. The authors have achieved the highest AP (0.3518) with Faster RCNN and Inception ResNet V2 backbone.

Detector	**Feature Extractor**	**AP^{medium}**	**AP^{large}**
Faster RCNN	ResNet 101 (640×640)	**0.2675**	**0.3944**
	ResNet 101	0.2507	0.3817
SSD	ResNet 101 FPN (640×640)	**0.2976**	**0.4198**
	ResNet 101 FPN	0.3118	0.4207

Table 5.5: Comparison of the results of this thesis and the baseline results of ABOships [18]. Thesis results are in **bold**.

SeaShips Baseline Comparison

The authors of SeaShips [21] have similarly presented baseline detection results with different architectures. The results of this thesis are mostly in line with the base detection results from the SeaShips paper, although there are some differences. This thesis uses the publicly available subset of the SeaShips dataset [22], with only 7 000 images out of the total 31 455 images. As with ABOships, it is unclear what input sizes some of the baseline results use. For example, SSD with MobileNet backbone and 608×608 input size achieves an AP of 0.7950, which is close to the AP of 0.7740 achieved in this thesis. Faster RCNN shows a larger difference in the AP metric, with the baseline result being 0.9240 and this thesis achieving 0.7620. This is likely due to the different input sizes used in the baseline and this thesis.

Detector	**Feature Extractor**	**AP**
Faster RCNN	ResNet 50 (640×640)	**0.7517**
	ResNet 50	0.9165
Faster RCNN	ResNet 101 (640×640)	**0.7620**
	ResNet 101	0.9240
SSD	ResNet 101 FPN (640×640)	**0.7740**
	MobileNet (608×608)	0.7950

Table 5.6: Comparison of the results of this thesis and the baseline results of SeaShips [21]. Thesis results are in **bold**.

Future Work

Work following this thesis could focus on improving the performance of the detectors in challenging maritime scenarios. Sensor Fusion is a simple yet effective way to improve the performance of the detectors, especially in low-light conditions, as shown in [7]. The use of RGB and IR cameras together can result in a more robust vessel detection system.

Elevating the number of sensors of different types, such as RGB, IR, and radar, can quickly become computationally expensive. Haghbayan et al. [58] have proposed

an efficient sensor fusion architecture for object detection in maritime environments. They have fused radar, LiDAR, RGB and IR sensors together using a probabilistic data association method, achieving reliable object detection in the maritime context.

Another interesting area of future work is to use an inertial sensor in order to alleviate the problem of waves and vessel movement, if using an on-board camera. As an example, Bertozzi et al. [59] have used an inertial sensor to reduce the problem of miscalibrations in obstacle detection and classification in the automotive context. Similar approach could be useful in maritime vessel detection, when using cameras on-board a vessel.

Finally, the development and use of even larger and more diverse maritime datasets would be beneficial for building a robust and general vessel detection system.

6 Conclusion

This thesis has presented relevant theory (Chapters 1, 2 & 3) on the topic of maritime vessel detection using deep neural networks, and implemented an experimental study (Chapter 4) with the aim of understanding the effect of different object detection architectures, the choice of backbone networks, and the feasibility of different maritime datasets for developing a vessel detection system. The results of the experiment (Chapter 5) have shown that Transfer Learning is a viable approach for developing a well-performing vessel detection system, even on consumer-grade hardware.

Both one-stage (SSD FPN) and two-stage (Faster RCNN) detector architectures were used in the experiment, and shown to perform well in maritime vessel detection and to produce similar results. In the experiment, SSD FPN architecture was shown to perform slightly better than Faster RCNN, but the difference was not large enough to deem Faster RCNN as unsuitable for the task.

The choice of the backbone network size (ResNet) was shown to have an effect when using the Faster RCNN architecture, but no significant effect was observed when using the SSD FPN architecture. The results suggest that using a larger backbone network size is beneficial when using the Faster RCNN architecture, but not as much when using the SSD FPN architecture. Thus, the choice of the backbone network is to be considered when developing a vessel detection system.

The feasibility of different maritime datasets was also studied. The experiment

used two different maritime datasets, ABOships and SeaShips, with different characteristics. The performance of the vessel detection system was vastly different on each of the datasets, where SeaShips dataset produced much higher performance metrics than ABOships. This suggests that ABOships dataset is more difficult for the detectors, and as such should be considered in future studies to improve the understanding of the performance of vessel detection systems in maritime environment.

The results of the thesis are inline with the baseline results from the literature. The thesis has provided additional insight into the effect of different object detection architectures, the choice of backbone networks, and the challenges in vessel detection in the maritime context. These results support the implementation of a vessel detection system in real-world maritime applications, and provide further understanding of the challenges in the maritime context.

Future research could focus on improving the performance of vessel detection systems in maritime environment, by handling the challenging aspects of the maritime context. Sensor fusion and multi-modal sensor data could be used to improve the performance of vessel detection systems, especially in low visibility conditions. The results of this thesis have shown that development of a vessel detection system for real-world maritime applications is feasible, and that the performance of the system can be improved with further research and experimentation.

References

[1] N. A. Stanton, P. R. Chambers, and J. Piggott, "Situational awareness and safety", *Safety science*, vol. 39, no. 3, pp. 189–204, 2001.

[2] W. Dai, Y. Mao, R. Yuan, Y. Liu, X. Pu, and C. Li, "A novel detector based on convolution neural networks for multiscale sar ship detection in complex background", *Sensors*, vol. 20, no. 9, p. 2547, 2020.

[3] Z. Zou, K. Chen, Z. Shi, Y. Guo, and J. Ye, "Object detection in 20 years: A survey", *Proceedings of the IEEE*, 2023.

[4] D. Lu and Q. Weng, "A survey of image classification methods and techniques for improving classification performance", *International journal of Remote sensing*, vol. 28, no. 5, pp. 823–870, 2007.

[5] W. Elmenreich, "An introduction to sensor fusion", *Vienna University of Technology, Austria*, vol. 502, pp. 1–28, 2002.

[6] H. Heiselberg and A. Stateczny, *Remote sensing in vessel detection and navigation*, 2020.

[7] F. Farahnakian and J. Heikkonen, "Deep learning based multi-modal fusion architectures for maritime vessel detection", *Remote Sensing*, vol. 12, no. 16, p. 2509, 2020.

[8] M. R. Endsley, "Measurement of situation awareness in dynamic systems", *Human factors*, vol. 37, no. 1, pp. 65–84, 1995.

[9] N. Wawrzyniak, T. Hyla, and A. Popik, "Vessel detection and tracking method based on video surveillance", *Sensors*, vol. 19, no. 23, p. 5230, 2019.

[10] A. Van den Broek, R. Neef, P. Hanckmann, S. P. van Gosliga, and D. Van Halsema, "Improving maritime situational awareness by fusing sensor information and intelligence", in *14th International Conference on Information Fusion*, IEEE, 2011, pp. 1–8.

[11] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*. Springer, 2006, vol. 4.

[12] X.-Y. Zhang, C.-L. Liu, and C. Y. Suen, "Towards robust pattern recognition: A review", *Proceedings of the IEEE*, vol. 108, no. 6, pp. 894–922, 2020.

[13] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[14] D. Berrar *et al.*, *Cross-validation*. 2019.

[15] F. Zhuang, Z. Qi, K. Duan, *et al.*, "A comprehensive survey on transfer learning", *Proceedings of the IEEE*, vol. 109, no. 1, pp. 43–76, 2020.

[16] S. Puttemans, T. Callemein, and T. Goedemé, "Building robust industrial applicable object detection models using transfer learning and single pass deep learning architectures", *arXiv preprint arXiv:2007.04666*, 2020.

[17] F. Farahnakian, L. Zelioli, and J. Heikkonen, "Transfer learning for maritime vessel detection using deep neural networks", in *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, IEEE, 2021, pp. 1–6.

[18] B. Iancu, V. Soloviev, L. Zelioli, and J. Lilius, "Aboships—an inshore and offshore maritime vessel detection dataset with precise annotations", *Remote Sensing*, vol. 13, no. 5, p. 988, 2021.

[19] T.-Y. Lin, M. Maire, S. Belongie, *et al.*, "Microsoft coco: Common objects in context", in *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, Springer, 2014, pp. 740–755.

[20] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database", in *2009 IEEE conference on computer vision and pattern recognition*, Ieee, 2009, pp. 248–255.

[21] Z. Shao, W. Wu, Z. Wang, W. Du, and C. Li, "Seaships: A large-scale precisely annotated dataset for ship detection", *IEEE transactions on multimedia*, vol. 20, no. 10, pp. 2593–2604, 2018.

[22] Z. Shao, W. Wu, Z. Wang, W. Du, and C. Li, *SeaShips (7000)*, `http://www.lmars.whu.edu.cn/prof_web/shaozhenfeng/datasets/SeaShips(7000).zip`, Accessed: 2023-11-07.

[23] S. Haykin, *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1998.

[24] L. Torrey and J. Shavlik, "Transfer learning", in *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, IGI global, 2010, pp. 242–264.

[25] N. Buduma, N. Buduma, and J. Papa, *Fundamentals of deep learning*. O'Reilly Media, Inc., 2022.

[26] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain.", *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[27] S. R. Dubey, S. K. Singh, and B. B. Chaudhuri, "Activation functions in deep learning: A comprehensive survey and benchmark", *Neurocomputing*, 2022.

[28] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks", *Advances in neural information processing systems*, vol. 25, 2012.

[29] J. Schmidhuber, "Annotated history of modern ai and deep learning", *arXiv preprint arXiv:2212.11279*, 2022.

[30] F. Rosenblatt, *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms* (Cornell Aeronautical Laboratory. Report no. VG-1196-G-8). Spartan Books, 1962.

[31] D. Svozil, V. Kvasnicka, and J. Pospichal, "Introduction to multi-layer feed-forward neural networks", *Chemometrics and intelligent laboratory systems*, vol. 39, no. 1, pp. 43–62, 1997.

[32] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, "Dive into deep learning", *arXiv preprint arXiv:2106.11342*, 2021.

[33] Z.-Q. Zhao, P. Zheng, S.-t. Xu, and X. Wu, "Object detection with deep learning: A review", *IEEE transactions on neural networks and learning systems*, vol. 30, no. 11, pp. 3212–3232, 2019.

[34] A. Dhillon and G. K. Verma, "Convolutional neural network: A review of models, methodologies and applications to object detection", *Progress in Artificial Intelligence*, vol. 9, no. 2, pp. 85–112, 2020.

[35] D. J. MacKay, *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[36] R. Girshick, "Fast r-cnn", in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[37] B. T. Polyak, "Some methods of speeding up the convergence of iteration methods", *Ussr computational mathematics and mathematical physics*, vol. 4, no. 5, pp. 1–17, 1964.

[38] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization", *arXiv preprint arXiv:1412.6980*, 2014.

[39] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization.", *Journal of machine learning research*, vol. 12, no. 7, 2011.

[40] T. Tieleman, G. Hinton, *et al.*, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude", *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012.

[41] W. Zhiqiang and L. Jun, "A review of object detection based on convolutional neural network", in *2017 36th Chinese control conference (CCC)*, IEEE, 2017, pp. 11 104–11 109.

[42] R. Szeliski, *Computer vision: algorithms and applications*. Springer Nature, 2022.

[43] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[44] J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders, "Selective search for object recognition", *International journal of computer vision*, vol. 104, pp. 154–171, 2013.

[45] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks", *Advances in neural information processing systems*, vol. 28, 2015.

[46] W. Liu, D. Anguelov, D. Erhan, *et al.*, "Ssd: Single shot multibox detector", in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The*

Netherlands, October 11–14, 2016, Proceedings, Part I 14, Springer, 2016, pp. 21–37.

[47] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition", *arXiv preprint arXiv:1409.1556*, 2014.

[48] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[49] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2117–2125.

[50] R. Padilla, S. L. Netto, and E. A. Da Silva, "A survey on performance metrics for object-detection algorithms", in *2020 international conference on systems, signals and image processing (IWSSIP)*, IEEE, 2020, pp. 237–242.

[51] T. Fawcett, "An introduction to roc analysis", *Pattern recognition letters*, vol. 27, no. 8, pp. 861–874, 2006.

[52] *COCO Detection Evaluation*, https://cocodataset.org, Accessed: 2023-12-14.

[53] *TensorFlow 2 Detection Model Zoo*, `https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md`, Accessed: 2023-10-31.

[54] *TFRecord*, `https://www.tensorflow.org/tutorials/load_data/tfrecord`, Accessed: 2023-11-15.

[55] A. Krizhevsky, "One weird trick for parallelizing convolutional neural networks", *arXiv preprint arXiv:1404.5997*, 2014.

[56] J. Huang, V. Rathod, C. Sun, *et al.*, "Speed/accuracy trade-offs for modern convolutional object detectors", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7310–7311.

[57] *TensorBoard*, `https://github.com/tensorflow/tensorboard`, Accessed: 2023-11-15.

[58] M.-H. Haghbayan, F. Farahnakian, J. Poikonen, *et al.*, "An efficient multi-sensor fusion approach for object detection in maritime environments", in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, IEEE, 2018, pp. 2163–2170.

[59] M. Bertozzi, L. Bombini, P. Cerri, P. Medici, P. C. Antonello, and M. Miglietta, "Obstacle detection and classification fusing radar and vision", in *2008 IEEE Intelligent Vehicles Symposium*, IEEE, 2008, pp. 608–613.