Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Lars Mansåker Angelsen

Experiments on the Mask RCNN Architecture with Synthetic Maritime Datasets

Improving Visual Perception through Heading Estimation, XAI, and Sensor Fusion

Master's thesis in Cybernetics and Robotics
Supervisor: Anastasios Lekkas
June 2020

NTNU
Norwegian University of Science and Technology

Lars Mansåker Angelsen

Experiments on the Mask RCNN Architecture with Synthetic Maritime Datasets

Improving Visual Perception through Heading Estimation, XAI, and Sensor Fusion

Master's thesis in Cybernetics and Robotics
Supervisor: Anastasios Lekkas
June 2020

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Summary

This thesis expands the functionality of the instance segmentation architecture Mask RCNN by including a heading estimation module. Later, modified versions of the Integrated Gradients and LIME methods are used to generate feature attributions for the heading predictions. The third experiment tries to improve the architecture's performance by including depth information along with its normal visual input. These experiments are motivated by the desire to improve the situational awareness offered by computer vision systems when used in maritime applications. The thesis can be summarized in four points.

The continued development of the synthetic maritime dataset generation software lays the foundation for the three experiments in this thesis. The final version can generate $\sim$ 600 synthetic samples per hour without the need for human labour. The samples include pixel-accurate depth maps, which can be used to simulate various sensor systems, along with more information about the objects in the image. The synthetic datasets featured in the experiments were created using 150 3D models of maritime vessels and consisted of 22000 samples.

The first experiment explores how the Mask RCNN architecture can be modified to predict the heading of detected objects in addition to its normal predictions. Three methods are implemented and compared to each other. One that predicts the heading as a single unit vector, one that predicts one heading vector per defined class and later selects the vector corresponding to the classification result, and one that reformulates the problem to a set of classification problems and produces the final heading through the mean shift clustering algorithm. During testing, the method based on the classification reformulation achieved the highest performance with a Median Absolute Error (medianAE) of 10.46°. When compared to humans (N = 29), the three models performed within the bounds of human performance.

The second experiment presents two modified feature attribution algorithms, based on Integrated Gradients and Local Interpretable Model-agnostic Explanations (LIME). These perform feature attributions for the heading predictions made by the heading estimation method utilizing a single vector. These methods are first validated on a simple toy dataset, on which they both perform fine, before they are used on heading predictions made for the synthetic maritime dataset. In this last application, the LIME based method performs notably worse than the Integrated Gradients based method. The feature attributions indicate that the Mask RCNN model uses a mix of highly semantic features and a few more basic features in its predictions.

The third and final experiment checks how access to depth information impacts the performance of the Mask RCNN architecture by passing depth maps as a fourth image channel along with the preexisting Red Green Blue (RGB) channels. It also implements a learnable depth-information fusion technique in which the depth maps are gradually combined with the feature maps internally in the feature extractor of the Mask RCNN model. In the end, neither of the modifications yielded an appreciable performance increase.

This thesis furthers the work started in *Explainability of Instance Segmentation Models Trained on Synthetic Datasets*[1], the project thesis from 2019 preceding this master's thesis.

Sammendrag

Denne oppgaven utvider funksjonaliteten til instans-segmenterings arkitekturen ”Mask RCNN” ved å inkludere en modul for retningsestimering. Senere blir modifiserte versioner av ”Integrated Gradients” og LIME brukt til å generere kjennetegn-attribusjoner for retningsestimatene. Det tredje eksperimentet forsøker å øke arkitekturens ytelse ved å inkludere dybdeinformasjon sammen med bildene. Disse eksperimentene er motivert av et ønske om å forbedre situasjonsforståelsen tilbudt av datasyn-systemer brukt i maritime applikasjoner. Denne oppgaven kan bli oppsummert i fire punkter.

Den fortsatte utviklingen av programmet for syntetisk marimt datasett generering legger fundamentet for de tre eksperimentene i denne oppgaven. Den endelige versjonen kan generere ~ 600 syntetiske bilder i timen uten å behøve menneskelig arbeid. Bildene inkluderer nå dybdekart nøyaktige på pikselnivå, som kan brukes til å simulere forskjellige sensorsystemer, samt mer informasjon om objektene i bildet. Det syntetiske datasettet brukt i eksperimentene var generert med 150 3D modeller av maritime fartøy og besto av 22000 bilder.

Det første eksperimentet basert på det syntetiske datasettet diskutert overfor utforsker hvordan ”Mask RCNN”-arkitekturen kan bli modifisert for å predikere retningen på oppdagede objekter i tillegg til dens normale prediksjoner. Tre metoder blir implementert og sammenlignet med hverandre. En som predikerer retningen som en enslig enhetsvektor, en som predikerer en enhetsvektor per definerte klasse og senere velger den som passer med klassifikasjonsresultatet, og en som omformulerer problemet til et sett med klassifikasjonsproblerer og senere produserer den endelige retningen gjennom en ”mean shift”-grupperingsalgoritme. Under testingen oppnådde metoden basert på klassifikasjonsomformuleringen den høyeste ytelsen med en medianAE på 10.46°. Sammenlignet med mennesker (N = 29), presterer de tre modellene på et tilsvarende nivå.

Det andre eksperimentet presenterer to modifiserte algorimer for kjennetegn-attribusjon, basert på ”Integrated Gradients” og LIME. Disse utfører kjennetegn-attribusjon for retningsprediksjoner laget av metoden som produserer en enslig enhetsvektor. Disse metodene blir første validert på et enkelt datasett, som begge yter bra på, før de blir brukt til på prediksjoner laget for det syntetiske maritime datasettet. I denne siste applikasjonen oppnår metoden basert på LIME merkbart dårligere ytelse enn metoden basert på ”Integrated Gradients”. Kjennetegn-attribusjonen indikerer at Mask RCNN modellen bruker en miks av kjennetegn med høy semantisk verdi samt et par med lavere semantisk verdi i sine prediksjoner.

Det tredje og siste eksperimentet sjekker hvordan tilgang til dybdeinformasjon påvirker ytelsen til Mask RCNN arkitekturen gjennom å inkludere dybdekart som en fjerde bildekanal i tillegg til de eksisterende RGB-kanalene. Eksperimentet implementerer også en lærbar teknikk for inklusjon av dybdeinformasjon hvor dybdekartene blir gradvis kombinert med nettverket internt i ”Mask RCNN”-modellen. Ingen av de to metodene med tilgang til dybdeinformasjon oppnådde en særlig bedre ytelse enn modellen uten tilgang.

Denne oppgaven fortsetter arbeidet i *Explainability of Instance Segmentation Models Trained on Synthetic Datasets*[1], prosjektoppgaven fra 2019 som er forgjengeren til denne masteroppgaven.

Preface

This thesis concludes my work on synthetic training data generation, the Mask RCNN architecture, and explainable artificial intelligence. It continues and expands upon my project thesis written in the Fall of 2019. The project has been performed in cooperation with Kongsberg Digital who provided me with software and hardware, as well as valuable guidance. My supervisor was Anastasios Lekkas, an associate professor of autonomous systems at the Department of Engineering Cybernetics. This thesis is my final work before graduating from NTNU with a master's degree within Cybernetics and Robotics, and represents the end of my five years as a student.

The following hardware was provided to me for the thesis:

- A Dell XPS 15 laptop, provided by Kongsberg. It was used for most of the programming work and for generating the synthetic datasets.
- A workstation equipped with a NVIDIA 1080ti graphics card, also provided by Kongsberg. It was used to train the Mask RCNN models and performed most of the explainable artificial intelligence experiments.

The following software resources were used:

- The Kongsberg Cogs graphics engine.
- 3D models supplied by Kongsberg.
- Matterport's Mask RCNN implementation[2].
- Ankurtaly's Integrated Gradients implementation[3].
- Marcotcr's LIME implementation[4].
- Rafaelpaddila's object detection metrics implementation[5].
- Scikit-image. For image processing in Python.
- OpenCV-Python. For image and video processing in Python.
- Matplotlib. For plotting in Python.
- Tkinter. For the GUI in the heading survey program.
- Pyinstaller. To 'compile' the survey program into an .exe file.
- Draw.IO. To create figures and diagrams.

Several people deserve my thanks for their help during this work and my time at NTNU. I want to thank my parents, Eli Mansåker and Thune Angelsen, and my younger brothers Sigurd M. Angelsen and Torvald M. Angelsen, as well as my girlfriend Ronja Björklund for their support during my time in Trondheim. From the Department of Engineering Cybernetics I want to thank Anastatios Lekkas for his guidance. From Kongsberg I want to thank Thorvald Grindstad and Jostein Bø

Fløystad for organizing the project, and Christopher Dyken for helping me with the Cogs program used in the synthetic dataset generation. Finally I'd like to thank everyone who participated in the manual heading estimation survey.

As this thesis is a continuation on the project thesis some parts of the text has been reused, while other parts have been written from scratch. What follows is a brief overview intended to make the job easier for the evaluator.

- Chapter 1 has been expanded and rewritten to motivate the new experiments performed in this thesis. Some sections are reused.
- Chapter 2 has undergone massive re-writings and expansions to facilitate more detailed explanations. Still, some parts have been reused from the project thesis.
- Chapter 3 is new.
- Chapter 4 contains much of the same discussions as the project thesis. The section discussing discriminative attributions is new, as are the sections describing the feature attribution for heading estimation.
- Chapter 5 has been slightly rewritten and expanded to contain the modifications necessary for the master's thesis. The comparison between K-Sim and Cogs has been removed. Much of the text has been reused.
- Chapter 6, 7, and 8 are new in this work. Some figures or explanations might still be taken from the project thesis. When this occurs the text will notify the reader.

As I write this, I am in the process of moving into an apartment in my childhood town Tønsberg. When thinking back to my time at NTNU in Trondheim, I am reminded of both good and tough times. In time, I suspect that only the good memories will remain. Goodbye for now Trondheim.

Table of Contents

List of Figures

Abbreviations

AI Artificial Intelligence. 3, 58–60

AIS Automatic Identification System. 46

ANN Artificial Neural Network. 14–16, 18, 19

AP Average Precision. 34

ARPA Automatic Radar Plotting Aid. 46

CNN Convolutional Neural Net. 12, 16, 17, 19, 20, 25, 32

COCO Common Objects in Context. 35, 87, 131, 134

FC Fully Connected. 15, 16, 19, 22, 32, 53

FCN Fully Convolutional Network. 20, 22, 24

FPN Feature Pyramid Network. ix, 24, 25

GAN Generative Adverserial Network. 22, 31

GNSS Global Navigation Satellite Systems. 42, 43

GPU Graphical Processing Unit. 12

HOG Histogram of Oriented Gradients. ix, 18

HSB Hue Saturation Brighness. 42

HSV Hue Saturation Value. x, 42, 81

IMU Inertial Measurement Unit. 42, 43

IOU Intersection Over Union. x, xiii, 35, 108

LIDAR Light Detection and Ranging. 8, 43, 95

LIME Local Interpretable Model-agnostic Explanations. i, ii, xii, xiii, 3, 5, 59, 65, 71–73, 75, 76, 92–95, 115, 117–121, 123, 126, 129, 134, 135

mAP Mean Average Precision. xiii, xiv, 98, 108, 130

MASS Maritime Autonomous Surface Ship. 3, 8

meanAE Mean Absolute Error. 54, 88, 100, 131

medianAE Median Absolute Error. i, ii, 54, 88, 113, 131

ML Machine Learning. 63–65

NLP Natural Language Processing. 30

RADAR Radio Detection and Ranging. 8, 43, 95, 96

ReLU Rectified Linear Unit. 15

RGB Red Green Blue. i, ii, 16, 40–42, 66, 68–70, 80, 96, 98, 135

ROI Region Of Interest. xii, 19, 20, 92, 93

RPN Region Proposal Network. xii, 20, 25, 87, 93

SLIC Simple Linear Iterative Clustering. 119

SVM Support Vector Machine. 18, 19

VOC2012 Visual Object Classes Challenge 2012. 19

XAI Explainable Artificial Intelligence. xi, xii, 3–6, 57–62, 64, 65, 67, 72, 92–94, 110, 115, 119, 121, 126, 135

Chapter 1

Introduction

1.1 Background and Motivation

Computer vision is expected to be one of the enabling technologies for autonomous vessels. However, within the application of this technology, which allows computers to extract information from the visual world, several challenges remain unsolved before the technology is ready for use in practice. Computer vision is in many ways a mature technology, but applications of state-of-the-art methods to safety-critical problems have revealed new requirements. Especially the interpretability of the systems have proven to be a challenge, since their complexity often prevents them from being fully understood through inspection.

The complexity of the visual world has led to the development of complex computer vision algorithms. Most state of the art methods utilize a machine learning method called Deep Learning. In simple terms Deep Learning utilizes large amounts of data to train networks of artificial neurons to perform a desired task. These networks, which are loosely inspired by biological brains, can contain several million artificial neurons. The learning part of Deep Learning varies between implementations, but a method often used is called supervised learning, and is based on large datasets of samples demonstrating an input-output relationship. For example: an input image and the desired output classification. The connections between the artificial neurons are then systematically strengthened or weakened through a method called backpropagation, which uses the gradient of a defined loss metric with respect to the individual network parameter to decide which connections should be strengthened and which should be weakened. This process is repeated numerous times during the training process and can result in systems with remarkable performance.

Even though Deep Learning based methods have been hugely successful in the last decade and have led to drastic improvements in computer vision systems, they have some downsides. To start, they often require large amounts of labelled training data for their training to be effective. Often this training data must be manually collected and labelled, which can be a time consuming task. In many machine learning projects, collecting and labelling the training data is one of the most expensive parts of the project. It is also crucial to collect good training data, or the final system might suffer from unintended biases or just not perform as well as required. Deep learning methods also suffer from a lack of intepretability, their huge complexity in addition to their use of large amounts of data in their training phase has made it difficult for humans to relate to them as anything but a black box. As a side effect of this, Deep Learning based systems are often described by their general architectures and training procedures, and not by the actual decision process of the system.

As information propagates through a network of artificial neurons, the neurons gradually expose information of higher semantic value. This highly semantic information is then used later in the network to perform the desired tasks. When performing instance segmentation, the network tries to detect and classify the objects visible in the image according to a set of predefined classes and create a mask covering the visible parts of each object. However, a trained human observer can extract much more information from a visual scene, for example the relative orientation or relative movement of objects. Creating computer vision systems capable of extracting more information from visual scenes would surely prove useful, either as the primary system for situational awareness, as an input to a larger system, or as a backup system. In maritime situations the relative headings of detected objects are often crucial for decision making. Therefore, estimating the relative headings of detected vessels through visual features is examined in this thesis. Previous works, such as [33], have

examined this problem before, but only in the case of single instance heading regression, and not in the more complex instance segmentation application. Tesla Incorporated, the American car manufacturer, is at the time of writing one of the leading developers of vision-based situational awareness for autonomous vehicles. Their Deep Learning based methods utilize cameras placed around the vehicle to create an internal situational awareness map which forms the basis for their autonomous driving policy[34].

In safety-critical applications of Deep Learning, such as in Maritime Autonomous Surface Ship (MASS) systems, this lack of interpretability represents a serious safety risk. A malfunction within navigation, control, or situational awareness can have dire consequences and naturally the computer systems that perform these tasks need to be as robust and interpretable as possible. Like the vessels themselves[35][36], the legislation surrounding MASSs is still being developed, but it will likely contain several requirements regarding the interpretability of the Artificial Intelligence (AI) systems used. This makes XAI an important enabling technology for autonomous ships.

Applying XAI to computer vision systems such as instance segmentation algorithms can mean that the XAI system highlights which parts of the input image led to a certain prediction. This process is called feature attribution, and can, in the case of a classifier model, be used to validate that the model has actually understood the general traits of some desired class, and is not just basing its predictions upon some bias in the training dataset. Several previous works have made progress in this field. Methods such as Integrated Gradients[32] and LIME[27] have both illustrated successful feature attribution and will be examined in this thesis. In theory, enabling AI systems to explain their inner workings opens many possibilities besides just model validation. By examining the AI's explanations a creator can gain insights into the strengths and weaknesses of the model and use this knowledge to further improve it. This process can lead to a better performing and more stable AI system, through the system guiding its own development. By allowing humans to learn from advanced AI systems, XAI could also lead to fundamentally new discoveries in many fields. In the far future, XAI might even become the connection between human beings and advanced artificial general intelligence systems.

Using simulated environments to train AI systems has become popular in the last few years. Simulated environments allow for a massive increase in training speed, by utilizing the computing power and parallelism offered by virtual environments. To illustrate; Waymo, the self-driving car company owned by Google, announced in July 2019 that their cars had driven 10 billion miles in virtual environments[37]. Using virtual environments to create datasets for computer vision algorithms has been explored in previous works, such as [38] which used the maritime simulator K-Sim to train an instance segmentation system. To enable training and analysis of machine learning systems, Kongsberg Digital wants to use Cogs, their 3D visualization engine for quick generation of labelled training data. Cogs promises to provide rapid 3D scene generation and improved flexibility with regards to ship selection, weather, background terrain, etc.

Autonomous maritime vessels in safety-critical applications are likely to employ a wide array of sensors for analysing their environments. In order to be effective, the information from these sensors must be combined into a single representation of the vessel's environment. There are infinitely many ways of achieving this, and determining the optimal way can be difficult. Synthetic training

data might be a good way to experiment with sensor fusion methods because of its cheap cost of dataset acquisition, high configurability, and ability to simulate a wide variety of sensor systems. Because of this, synthetic training data might allow experiments that would otherwise be infeasible, either because of the prohibitive cost of collecting the datasets or difficulty in correctly labeling the samples. Experiments with synthetic datasets could therefore be used to determine whether it would be a good idea to implement a machine learning system in the real world.

Synthetic dataset generation also allows the construction of ”what if?” experiments because of the high fidelity information that can be extracted from the synthetic dataset generators. Experiments such as these can, for instance, give developers an indication of the maximum performance increase to be expected from a machine learning system given an improvement in sensor technology. This thesis explores one such ”what if?” question. Specifically, how much would the performance of the Mask RCNN architecture increase if the network had access to a pixel perfect depth-map describing the visual scene? Currently no sensor system offers such performance, either being limited by operational range, or resolution, as is the case with LIDAR and RADAR. But it would be interesting to observe how the performance of the Mask RCNN based systems would change if this information was accessible.

Synthetic datasets also give fine control to the creator, which enables precise comparisons and analysis. An example of synthetic datasets being used to facilitate comparisons and analysis is [39] in which various methods for feature selection were compared on several different precisely controlled synthetic datasets. The computer vision systems developed in this thesis are analysed in much greater depth than in other works, thanks to the large amount of information available from the synthetic datasets generated in this thesis. This is expected to provide the reader with a great deal of insight into the models developed.

1.2 Objectives

The thesis has the following problem description.

The main goal of this research project is to explore the feasibility and usefulness of using computer-graphic simulators (Cogs and K-sim) for development and validation of Explainable AI techniques in maritime scenarios. A secondary goal is to evaluate what information can be extracted from visual scenes by Deep Learning techniques. The work falls within the area of computer vision as an enabling technology for autonomous ships.

This can be simplified into four sub-objectives.

- Research the state of the art methods for computer vision and XAI systems.
- Modify the computer vision systems to extract more information from visual scenes.
- Construct a software framework for automatic generation of labelled training images with the Kongsberg Cogs 3D visualization engine.
- Use the generated training data to train Deep Learning based computer vision models.
- Examine the trained models with XAI methods.

1.3 Contributions

This work further improves on the synthetic dataset generation program implemented in [1]. The new version not only creates datasets with more information exposed to the end user, such as a pixel-accurate depth-maps and relative object headings, but also improves the quality of the generated datasets by ensuring that the objects are placed at distances which it would be realistic to expect that a given camera configuration could view.

The thesis also compares three different methods of estimating the relative heading of detected objects. All three methods are implemented as modifications to the existing Mask RCNN architecture. The first method adds a heading vector regression head to the existing Mask RCNN class/bounding box regression, this additional regression head is then trained with a smooth L1 loss function. The final heading angle is calculated with the atan2(y,x) function. The second method is similar to the first, but instead calculates one unit vector per defined class. The final unit vector is then sampled according to the class with the highest probability score and converted to an angle with the atan2(y, x) function. The heading regression is trained with a smooth L1 loss function. Method three reformulates the heading regression problem as a set of classification problems, before calculating the final heading using a mean shift algorithm. The classification layer is trained with a categorical cross-entropy loss function. The methods are modified versions of the ones presented in [33].

Two XAI methods are modified to generate feature attributions explaining the headings predicted by the first method discussed above. The first feature attribution method is Integrated Gradients, which repeatedly calculates the gradient of the output unit vector component values with respect to the input image while gradually transforming the image from a baseline to the actual input image. By evaluating which image pixels consistently have the highest gradients during this process it highlights the most influential input pixels with respect to the output values. In order to create feature attributions for the two-dimensional heading predictions, the method is modified to calculate the gradients relative to the two components of the predicted heading vector, which are merged via a merging function into a single image. The second feature attribution method is LIME. It trains an interpretable surrogate model on altered versions of the input image and highlights the most influential input image segments based on the interpretable surrogate model. When modified to create feature attributions for the heading predictions, the heading predictions generated during the training of the surrogate model are converted to pseudo-classification values ranging between 0 and 1, depending on how close to the original predicted heading the intermediate result is. The LIME method used for this task is specially adapted to the instance segmentation problem by only examining a cropped version of the total input image. In theory, this allows higher precision in the generated feature attributions.

The final experiment in this thesis analyses whether the inclusion of depth information along with the input image allows the Mask RCNN to produce more accurate predictions. Two methods for depth inclusion are implemented. One which simply appends the depth-map as a fourth image channel in the input image, and one which gradually merges the information through a learnable architecture. This experiment is only a proof of concept, and serves to illustrate the possibilities enabled by the expanded synthetic dataset generation. In future works, these synthetic depth-maps can be used to simulate depth sensors for experiments with sensor fusion.

1.4 Outline

This work is split into eight chapters. Chapter one presents the motivation behind the project, the remarkable performance offered by Deep Learning based methods, their unfortunate lack of interpretability, and the benefits offered by synthetically generated training data. Chapter one also highlights the project's contributions and outlines its structure. Chapter two introduces the theory required to understand the experiments and the results, mainly focusing on computer vision, machine learning, and Deep Learning. It also tries to develop a sense of intuition behind neural networks and their inner workings. Synthetic datasets and sensor fusion are also briefly covered. It is written with the intent of being understandable for a person with only surface level knowledge of computer science and mathematics.

Chapter three covers heading estimation. It discusses notable aspects of heading estimation, different formulations and solutions to the problem, and how the three heading estimation methods analysed in the experiments are implemented in the Mask RCNN architecture. Chapter four presents the field of XAI. It discusses how to create interpretable computer vision systems, how a good explanation is structured, and what it means for a machine learning system to be interpretable.

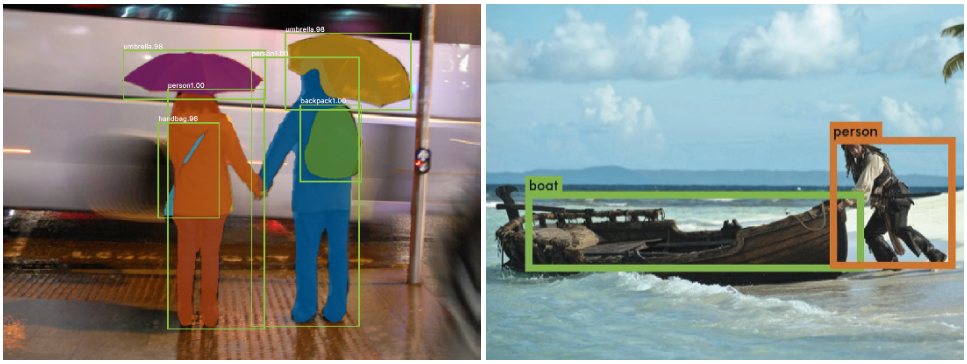
Chapter five presents the system for generating synthetic datasets with Kongsberg Cogs, what information is presented to the end user and various design aspects of the system. Chapter six introduces the experiments, as well as the experimental setup with which they were performed. Chapter seven presents and discusses the results of the aforementioned experiments. Chapter eight concludes the thesis and suggests improvements and ideas for further work.

Chapter 2

Theory

This chapter presents the relevant background theory needed to understand the thesis, the experiments, and the results. It begins by introducing the field of computer vision, the technology that allows computers to understand the visual world around them, and why it is useful for environmental awareness in autonomous vehicles. It then continues on to machine learning which is the field of study that allows computers to learn desired tasks and functions without being explicitly programmed. The chapter describes the different types of machine learning, while focusing on supervised learning, which is the method employed in this thesis.

Thereafter, the new field of Deep Learning is covered. This is the technology that has enabled the huge performance increase seen within state-of-the-art computer vision systems in the last decade. The section highlights the Deep Learning methods used in this thesis and discusses their defining features. The next topic is synthetic dataset generation. This section describes some guiding principles useful when creating programs that generate synthetic datasets and when training machine learning systems on the generated synthetic datasets. Finally sensor fusion is briefly covered, with the intent of motivating why it is so useful as well as highlighting some previous methods used to couple it with Deep Learning based methods.



(a) Illustration of instance segmentation, taken from [9]. Note that the individual pixels are labeled by class and instance.

(b) Illustration of object detection performed by Yolo v1. Note the generated bounding boxes as well as a class label. The figure is from [40].

Figure 2.1: Examples of computer vision tasks.

2.1 Computer Vision

Computer vision is a wide field with many different applications, everything from simple edge detection to more nuanced tasks like classifying different breeds of dogs fall within this field. Today computer vision is widely used in the industry, with applications such as quality control in assembly lines or visual inspection of existing systems. In recent times the performance of these systems have increased dramatically thanks to innovations within the fields of machine learning and Deep Learning. Computer vision systems can now do things previously thought impossible and is an active research area which is progressing rapidly.

Naturally, computer vision is an important component in autonomous vehicles. In modern autonomous systems cameras are one of several sensor systems used, often used in combination with Light Detection and Ranging (LIDAR), Radio Detection and Ranging (RADAR), and/or ultrasonic sensors. In maritime applications computer vision is used to detect other vessels as well as environmental obstacles. In MASS systems this information is then used in combination with other sensor data to form the situational awareness of the vessel. The situational awareness system handles deduced information about the movements and intentions of other vessels. From this information the behaviour of the vessel is defined. Because of this important role, the integrity of the computer vision system is critical for the continued operation of the vessel.

Depending on the role of the computer vision system, there are many tasks that the system can perform. Example tasks can be recognition of facial features, used as a safety feature in some phones, or vehicle counting, for traffic monitoring. For MASS systems the following tasks are the most relevant.

- Object detection. Detecting the existence of an object within an image. Often provides a bounding box around the object in addition to a class label. An example is shown in Figure 2.1b.

- Semantic segmentation. Labelling each pixel as one of a set of defined classes.
- Instance segmentation. Detecting all the objects present in an image, classifying the detected objects and generating a mask covering their silhouettes. An example is shown in Figure 2.1a.
- Panoptic segmentation. This is a combination of the two aforementioned tasks. All pixels in the image are assigned a class and an object instance, if it is part of one.

All the four tasks defined so far extract differing amounts of information from the visual scene. A system which implements object detection could, based on the location of the detected objects in the image, known camera parameters, and the predicted class, extract crude 3D positional information from the visual scene. The vessel's decision system could then use this information, along other sensor data, when making its decisions. If the computer vision system instead implemented semantic segmentation, more fine-grained information could be extracted from the visual scene. This could be more precise positional information from boats or the shoreline. However, semantic segmentation is not ideal, since it fails to separate different object instances. Semantic segmentation will not detect two boats if the boats are partially overlapping in the image. Instead it will predict a single group of boat pixels. Instance segmentation does not suffer from this effect and would be able to separate the two boats.

In this thesis, an instance segmentation (Mask RCNN) algorithm is applied to dense visual scenes. This means that if the system performs perfectly, all the pixels in the input image would be covered by object instances. Using the Mask RCNN architecture for this is not ideal and can lead to overlapping detections and other artifacts, since the implementations of instance segmentation used in this work do not facilitate communication between the separate instance detections. A problem caused by this is that the system would sometimes detect both ocean and a vessel at the same space in the image. On the other hand, a system that implements panoptic segmentation would not suffer from these artifacts, since it assures that each pixel can only be assigned to one instance/class.

2.2 Machine Learning

Creating self-learning computers has been viewed as sort of a holy grail within computer science. The differentiating feature between traditional algorithms and machine learning algorithms is that the computation within a machine learning algorithm does not have to be explicitly programmed. Instead it will be generated dynamically based on the provided data. Of course a programmer still has to define the framework that the machine can learn within. There are several frameworks to choose from, each with their own strengths and weaknesses. The three most common methodologies will be presented next.

The three main types of machine learning are:

- Supervised learning
- Unsupervised learning
- Reinforcement learning

Figure 2.2: The MNIST dataset is often used to benchmark machine learning algorithms. It consists of 70000 image-number pairs. The figure is from [6].

Since this thesis will implement supervised learning, unsupervised and reinforcement learning will only be covered superficially. When performing supervised learning the machine is provided with input-output pairs. The machine is then expected to learn the relationship between the different pairs. This relationship can be described mathematically as

$$Y = h(X) \tag{2.1}$$

where Y is the desired outputs, X is the input data, and $h(X)$ is their real world relationship. A machine learning algorithm approximates $h(X)$ with the function

$$Y \approx \hat{h}(X) \tag{2.2}$$

which, if the algorithm performs well is approximately equal to $h(X)$. This task is usually split into two types, classification and regression. When performing classification the input is to be assigned a class based on the input features. Normally the output is a probability distribution over the defined possible classes. An example application is classifying hand-written digits. A dataset often used for this purpose is the MNIST dataset. Some samples from the MNIST dataset is in Figure 2.2. The defined classes would in this case be the digits 0-9, and the input would be the images of the handwritten digits. When performing regression the model learns to approximate a function with a scalar or vector output. An example task could be to estimate the price of an apartment based on the number of bedrooms, floor area, etc. The input would then be a vector describing an apartment's properties, and the output value would be the price of the apartment. Naturally, the idea behind supervised learning is that once the model has been trained, it should be able to perform its task on examples it has never seen before.

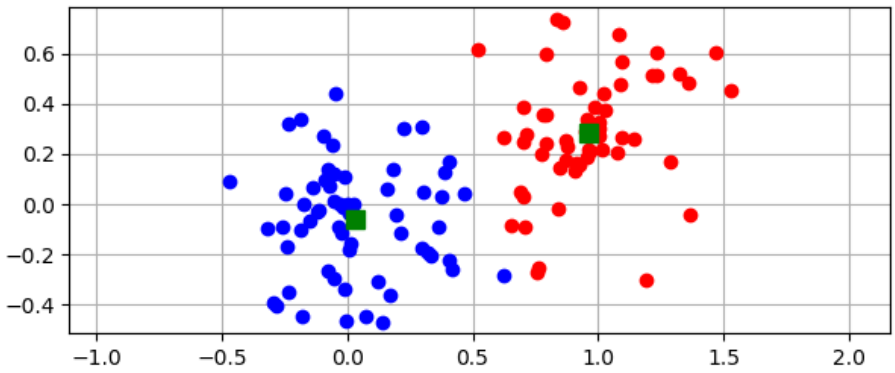


Figure 2.3: These datapoints have been clustered by the unsupervised learning algorithm k-means. The two different colors represent a different class. The figure was generated using a Python script and the Scipy package.

The dataset is typically divided into three parts.

- The training dataset, which is used to train the machine learning algorithm. It is the largest part, usually constituting about 90% of the total dataset.
- The validation dataset, which is used to tune the machine learning system's framework[41]. The values describing the machine learning framework is often called the hyperparameters of the machine learning system.
- The test dataset, which is used to test the performance of the final algorithm. The test dataset should represent the real world data as closely as possible.

There are several reasons for splitting up the dataset this way, one of them is to prevent overfitting. Overfitting occurs when the algorithm memorizes the dataset instead of learning its general traits, which will in turn degrade the algorithm's performance when applied in new situations. How well a machine learning system works on new data is described by how well the system generalizes. The problems surrounding overfitting will be explained further in Section 2.3.11.

Unsupervised learning, which is the second main type of machine learning, differs from supervised learning in that it does not require the desired outputs for learning. This means that the programmer doesn't have to provide the algorithm's desired answers. This reduces the work required to train the system. In supervised learning a substantial part of the work is spent labelling the datasets. For example: labelling the ship types present in the images in a maritime dataset. This is often done manually, but methods that do this automatically have begun to emerge. This thesis will demonstrate a method for automatic dataset generation. That means that not only is the desired answers generated automatically, but also the corresponding inputs. This further decreases the work needed in supervised machine learning. Unfortunately unsupervised machine learning methods are often limited to clustering and pattern detection in the data. One such unsupervised machine learning algorithm is the k-means algorithm, it is illustrated in Figure 2.3 where it clusters a set of points

based on their position in the image.

The final type of machine learning covered is reinforcement learning. This method does not require a dataset at all. Rather it is based upon agents and environments. The agents are placed in the environment and learns by trial and error. The programmer defines a reward function which rewards the agent if it acts in a desired manner and punishes it if it fails to do so. The agent tries to receive as much reward as possible by optimizing its strategies.[1] After repeated attempts the agent has (hopefully) created a successful strategy for maximizing the reward function. In recent years state-of-the-art reinforcement learning methods have improved dramatically and are now able to master complex problems demanding long term planning, such as the video game Starcraft 2[42]. A downside to current reinforcement learning methods is that the agents often require large numbers of attempts before they become prolific at a task. This has limited most current reinforcement learning projects to software environments where the training can be accelerated.

2.3 Deep Learning

The mathematics that form the backbone of Deep Learning have existed for some time. In essence it is simply repeated linear transformations in combination with nonlinear activation functions. The thing that has changed lately, and made Deep Learning the dominant type of machine learning, is the accessibility of hardware acceleration in the form of high performance computational units (Graphical Processing Units (GPUs)). These can speed up the calculations by several orders of magnitude and have enabled increasingly complex Deep Learning models. Innovations in network architectures have also led to higher performance by either improving the computational efficiency, like Convolutional Neural Nets (CNNs), or by improving the training procedure.

2.3.1 Perceptrons and Universality

In the 1950's and 1960's, a scientist by the name Frank Rosenblatt experimented with a new form of computational algorithm. It was inspired by the neurons in animal brains and could, similarly to most animals, learn from past experiences. He called this new algorithm the perceptron[43]. In a research paper published in 1957, Rosenblatt defined the perceptron as an electronic or electromechanical system which learns to recognize similarities or identities between patterns of optical, electrical, or tonal information, in a manner which may be closely analogous to the perceptual processes of a biological brain[44]. The perceptrons, similarly to the artificial neurons used in modern artificial neural networks, perform mathematical operations on a set of inputs and produce an output. More specifically they first perform a weighted sum of the inputs values, which must have a binary value, and outputs the binary result of a threshold operation on this weighted sum. This mathematical operation can be written as

$$output = \begin{cases} 1, & \text{if } \sum_{i=0}^{N} w_i x_i + b \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.3}$$

[1] The strategy is often referred to as policy.

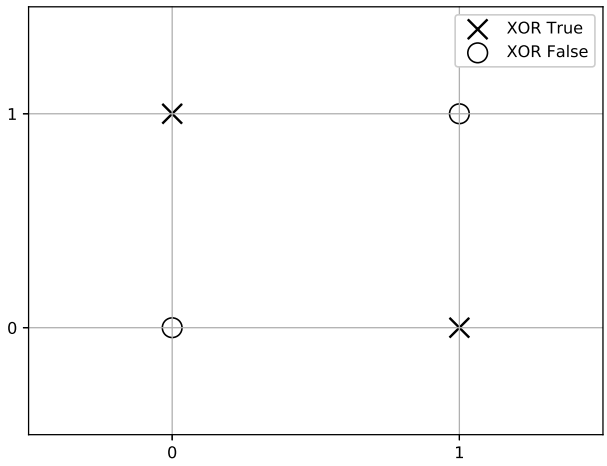


Figure 2.4: XOR function values.

where w_i is the weight corresponding to the ith input, x_i is the ith binary input, and b is the bias. The bias can be interpreted as the threshold value in the threshold operation. As will become clear in the following sections this mathematical operation is very similar to the the one used in modern artificial neural networks. In the same paper from 1957 he also describes a learning method in which connections that are active while the network outputs correct results are strengthened and proposes that by training the network over many samples of data the networks performance will increase. This is reminiscent of how the modern method of training artificial neural networks operate. The most popular modern training procedure is gradient descent with backpropagation, and is described in Section 2.3.9.

Early experiments with perceptrons quickly unveiled that single layer perceptrons are severely limited in which tasks they can perform. It turns out single layer perceptrons can only successfully classify linearly separable data, given by functions like the AND function or the OR function. Non-linear functions such as the XOR function have proven impossible for single layer perceptrons to learn. This was first shown in the book *Perceptrons*[45] which released in 1969. A simple mathematical analysis backs up this claim. From Equation 2.3 it is clear that the single layer perceptron algorithm classifies based on the linear hyperplane defined by

$$\sum_{i=0}^{N} w_i x_i + b = 0 \tag{2.4}$$

where w_i is the weight corresponding to the ith input, x_i is the ith binary input, and b is the bias. In the two dimensional case this hyperplane exists as a line. Through visual inspection, it is clear that the XOR function, shown in Figure 2.4, can't be correctly modelled by a single layer perceptron since it is forced to classify via linear hyperplanes, or in this case: a line. The development of

multi-layer perceptron networks aimed to alleviate this issue. In fact the Universal Approximation Theorem states that a feed-forward network with a single hidden layer, which implies a network of two layers of artificial neurons or perceptrons, containing a finite number of neurons can approximate continuous functions on compact subsets of R^n, under mild assumptions on the activation function. Various versions of the proof have been presented, in 1989 it was proved for the sigmoid activation function[46] and in 1993 it was proven for all non-polynomial activation functions[47]. However, just because a network with a single hidden layer is theoretically able to approximate any continuous function doesn't mean that one hidden layer is enough for real world applications. The Universal Approximation Theorem only states that the number of required neurons is finite, not the system can be implemented within today's hardware constraints, or that a one-layer solution is best.

2.3.2 Artificial Neural Networks

Modern Deep Learning is based upon Artificial Neural Networks (ANNs) which are networks of artificial neurons, a further development of the perceptron. Similarly to the perceptron, artificial neurons are mathematical constructs with several inputs and one output. It performs two operations on the inputs. The first operation is a weighted sum of the input values. The only difference from the perceptron is that it accepts continuous input values. As with the perceptron, this sum includes a bias value. The next operation performed on the input is the application of a non-linear activation function. The activation function is what enables neural network to exhibit non-linear behaviour. Without it the network would be limited to modelling linear systems. See Figure 2.5a for an illustration of such an artificial neuron. The two operations performed by an artificial neuron can be written as

$$z = \sum_{i=0}^{N} w_i x_i + b \tag{2.5}$$

and

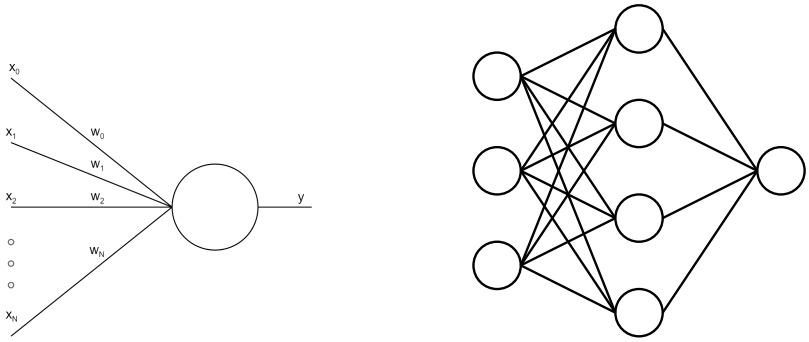
$$y = \phi(z) \tag{2.6}$$

where ϕ is the non-linear activation function, x_i is an input, w_i is the weight corresponding to that input, and b is the bias. z is used as a intermediate variable to represent the weighted sum and facilitate easier explanation.

There are several different activation functions in use today, each with their own strengths and weaknesses. A few years ago the most popular one was the sigmoid function (2.7).

$$\phi(z) = \frac{1}{1 + e^{-z}} \tag{2.7}$$

However, the sigmoid function has several downsides. For one it suffers from a phenomena called gradient saturation, which occurs for large positive or negative values of z. For such values, the gradient of the sigmoid function becomes very small. Since, when performing gradient descent, the training speed is proportional to the gradient of the activation function, this small gradient value causes slow learning; especially when the gradient is propagated through several layers using the



(a) An artificial neuron with inputs, weights, and output shown.

(b) A network of artificial neurons, the information propagates from the input neurons(left) to the output neurons.

Figure 2.5: The building blocks of a neural network.

sigmoid activation function. The function is also computationally complex, which means that each training step and inference step takes longer to calculate. Since then, another activation function called Rectified Linear Unit (ReLU) (2.8) has replaced the sigmoid in many applications. Compared to the sigmoid activation function, it offers several improvements. For one, it is much more computationally efficient than the sigmoid activation function, as indicated by Equation 2.8. Another benefit is that its derivative is either 0 or 1 which simplifies the calculations required for backpropagation. However, neurons using the ReLU activation function can "die" if their weighted averages are zero for all inputs, as this would cause their gradients to be zero which would stop the learning process. Modified ReLU functions, such as Leaky ReLU[48], have been made to solve this issue. The training process is described further in Section 2.3.9.

$$\phi(z) = max(0, z) \tag{2.8}$$

In the final layer of a neural network which performs classification it is common to use the softmax activation function (2.9). It generates a probability distribution over all the layer outputs, ensuring $\sum_{i=0}^{N} z_i = 1$.

$$\phi(z) = \frac{e^{z_i}}{\sum_{i=0}^{N} e^{z_i}} \tag{2.9}$$

Artificial neurons like the ones described above are the building blocks of ANNs. When arranged in a layered structure, these simple operations combine to expose complex patterns in the input data. The early layers detect simple patters, such as edges (if the input is an image), while the later layers find features with more semantic information. This process of uncovering patterns of high semantic value is called feature extraction. In literature, these layers are often referred to as Fully Connected (FC) layers.

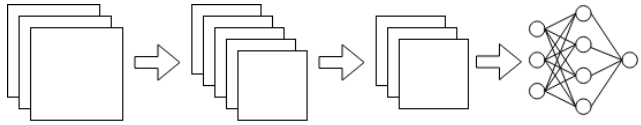


Figure 2.6: A convolutional neural network with two fully connected layers at the output for classification. The boxes represent feature maps, which are modified by kernel convolutions or pooling operations.

2.3.3 Convolutional Neural Networks

For computer vision applications another Deep Learning architecture, the Convolutional Neural Net (CNN), is commonly used. CNNs differ from traditional artificial neural networks in that instead of iteratively calculating weighted sums of the previous layer's neurons outputs, they calculate weighted convolutions. This allows them to incorporate spacial information, which is how detected features are positioned relatively to each other, in their calculations. For example: two edges perpendicular to each other create a corner, or if they are parallel, a ridge. In the learning process the CNN learns to extract information of higher and higher semantic value through the iterative application of its convolutional filters. To expand on the previous example: a filter deeper in the CNN might learn that two ridges in a certain constellation form a nose, a feature that would be very useful when detecting faces. Each layer of a CNN usually have several such convolutional filters that scan the previous layer's output for patterns. After several layers the outputs of the convolutional filters contain highly refined information extracted from the input image, these matrices of high semantic value are usually called feature maps. Another advantage of CNNs is that they reuse the network weights thanks to the repeated application of the same convolutions over the entire input image or feature map. This leads to a smaller memory footprint when compared to equivalent traditional ANNs. If a CNN is used for classification it is common that the final layers are FC layers. This is illustrated in Figure 2.6. In this configuration the CNN is used as a feature extractor, while the FC layers perform the classification task.

There are two operations that define CNNs. One is the aforementioned weighted convolution and the other is the pooling operation. The weighted convolution will be considered first. It is similar to the weighted sum in an artificial neuron but work over a patch of data instead of all the input data. This operation can be expressed as

$$G[m,n] = \phi((f * h)[m,n]) = \phi(\sum_j \sum_k h[j,k] f[m-j, n-k] + b) \tag{2.10}$$

where G is is the resulting matrix, f is the input matrix, ϕ is a non-linear activation function, b is the bias, and h is the learnable kernel[49]. Equation 2.10 shows the 2 dimensional example, but it can easily be expanded to three dimensions, which is needed if the input data has a depth layer, such as the three RGB layers in a color image. The kernel is a tensor, or a matrix in the case of a 2D input, that defines the weight applied to each part of the image patch. During the training process the values in the kernels and biases in the network are optimized.

The CNN operation is defined by a set of hyperparameters. These decide the size of the output matrix (W_{out}, H_{out}, D_{out}), the number of network parameters, and what kind of features the network's layers are able to learn. The parameters are:

- Kernel size. (F)
- Amount of padding. (P)
- Stride length. (S)
- Number of filters. (N)

$$W_{out} = \frac{W_{in} - F + 2P}{S} + 1 \qquad (2.11) \qquad H_{out} = \frac{H_{in} - F + 2P}{S} + 1 \qquad (2.12)$$

$$D_{out} = N \qquad (2.13) \qquad N_{parameters} = (F \times F \times D_{in} + 1) \times N \qquad (2.14)$$

If the kernel is bigger the network layer learns larger, more generic features, but also requires more storage space for the kernel parameters. Larger kernels also counteract the weight sharing principle that defines the CNN architecture. With all other things being equal, a larger convolutional kernel also leads to a decrease in the size of the output feature map, as seen in Equations 2.11 and 2.12. The amount of padding (P), can be used to preserve the spatial size of the feature maps through a convolutional layer. Zero padding is often used, this causes the the padded values to be set to zero. The stride length (S) decides how far the convolutional filter moves between each convolution. It can also be used to control the spatial size of the output feature map, since a large stride leads to a drastically smaller output feature map. By varying the size of the kernel and stride length it is also possible to induce an overlap between the convolutions, if that is desired.

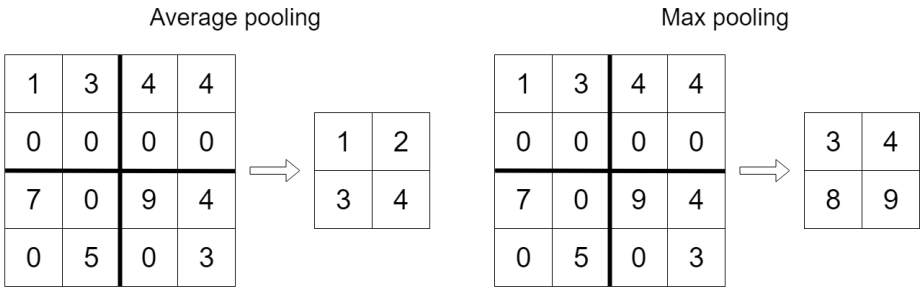


Figure 2.7: The pooling operations max pooling and average pooling. Here a 4x4 feature map is reduced to a 2x2 feature map.

The final operation that defines CNNs is the pooling operation. The idea behind the pooling operation is to reduce the size of the output feature maps while preserving the semantic information they contain through a downsampling method. For many applications the rest of the network only needs to know whether a feature exists in the image, and not detailed information regarding its location. There are several pooling methods, differentiated by how they calculate the values contained in the output feature map. The most common pooling operations are max pooling and average pooling. Max pooling outputs the maximum value present in a section of the input feature map, while average pooling outputs the average value. Max pooling and average pooling are showcased in Figure 2.7.

2.3.4 Hand Crafted Features vs. Learned Features

Traditional machine learning systems that perform classification tasks on visual inputs are often split into two parts, the feature extractor and the classifier. The job of the feature extractor is to transform the input image into a format suitable for the next component, the machine learning classifier. This can be by applying a simple edge detection algorithm or other more advanced transformations such as Histogram of Oriented Gradients (HOG). In traditional implementations the extracted features are then sent to a machine learning based classifier such as a Support Vector Machine (SVM). One potential advantage that ANNs have over traditional machine learning approaches such as HOG based classifiers is that, as described in the previous section, they learn both the feature extraction and classification in parallel. The early layers of the network learn simple filtering transformations, while subsequent layers extract information of higher semantic value from the data. This allows the classification in artificial neural networks to be based on features of higher semantic value than if it was based on feature descriptors designed by a human, since it is very difficult to hand-design a feature descriptor of high semantic value. It also allows the feature extractor and classifier to be finely tuned to each other.

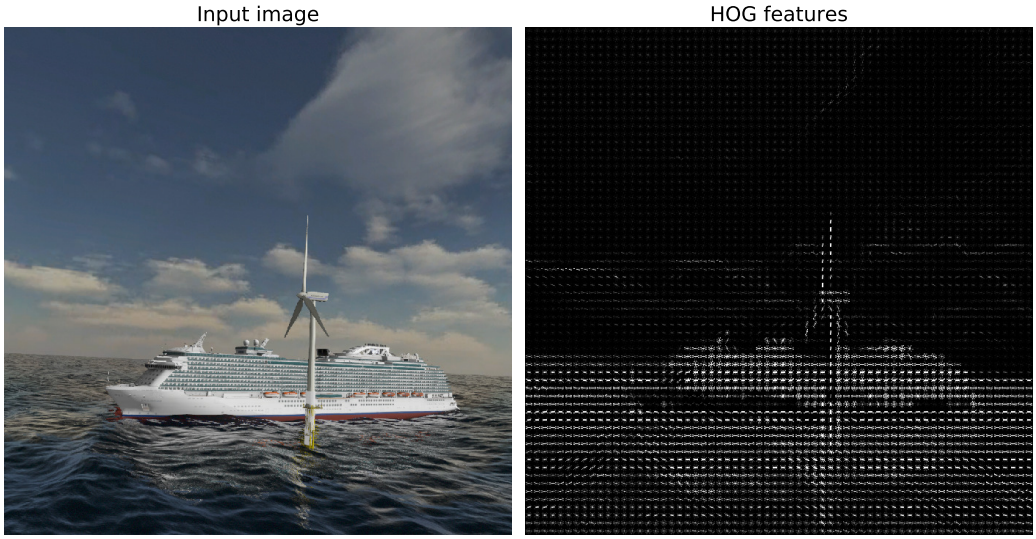


Figure 2.8: An example of the HOG feature extractor. The image is from the synthetic dataset generated later in the thesis, and the HOG feature descriptor is generated by the Python module scikit-image.

Most current human-designed feature extractors are based upon simple operations or transformations of the input image. Take the aforementioned HOG feature extractor which, like the name implies, is based upon creating histograms of the gradient-directions present in image patches. Figure 2.8 shows a feature descriptor generated by the HOG feature extractor. It is clear that this feature descriptor contains limited semantic information besides the gradient directions. A downside to the integration of the feature extraction and classification in artificial neural networks is that the system

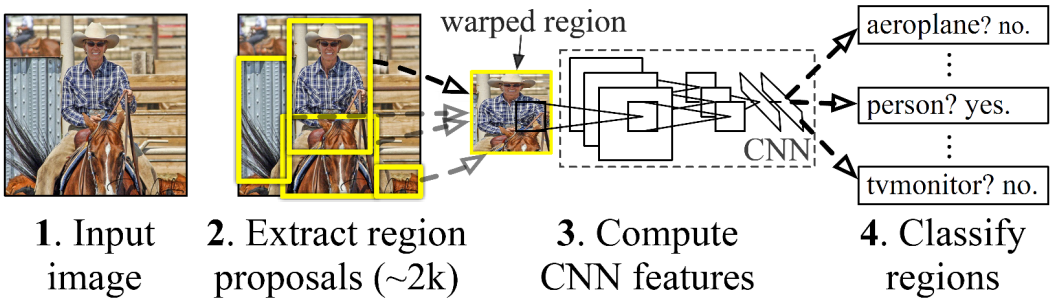


Figure 2.9: The RCNN pipeline. The region proposal step, the feature vector generation, and the linear classification is shown. The Figure is from [7].

becomes less interpretable, since the feature extractor no longer follows precisely defined rules created by a human programmer. How the interpretability of machine learning systems such as ANNs can be increased is discussed in Section 4.4.

2.3.5 Mask RCNN

In 2014 a research group from UC Berkeley published a new method for object detection. The method combined a CNN classifier with a preprocessing step called region proposal. The region proposal step found the most ”interesting” parts of the image, thus ensuring that the ensuing classifier could work as effectively as possible by only looking at the parts of the image that was expected to contain objects. They called this method RCNN (Regions with CNN features). Note that original RCNN architecture did not use FCs network for the final classification, opting instead to use a Support Vector Machine (SVM), which is a type of linear classifier[7]. It did however use two FC layers from the CNN feature maps to the feature vectors used in the SVM. The interesting areas of the input image were termed ROIs and were warped to fit the CNN feature extractor, which required fixed size inputs. The RCNN pipeline is shown in Figure 2.9. The RCNN method achieved the highest performance ever on the Visual Object Classes Challenge 2012 (VOC2012) dataset and received a lot of attention at the time. After some time improved versions were developed, such as the aptly named Fast RCNN and Faster RCNN architectures, which both improved on the RCNN architecture’s inference speed and prediction accuracy.

Fast RCNN, introduced in 2015, replaced the SVM classifier with a FC network for determining the class and performing bounding box regression[50]. In contrast, the original RCNN method simply used the bounding boxes from the region proposal step as object bounding boxes. By further refining the bounding boxes, the Fast RCNN architecture achieved more accurate bounding box predictions. Fast RCNN also extracted a feature map from the input image before it generated the feature vector for each ROI, this way the feature map was reused among the different detections. It also replaced the ROI warping in favor of ROI pooling layer. This was a natural replacement since the new ROIs were projected on a feature map instead of the input image. The ROI pooling layer is equivalent to the max pooling shown in Figure 2.7. These improvements, as well as some modifica-

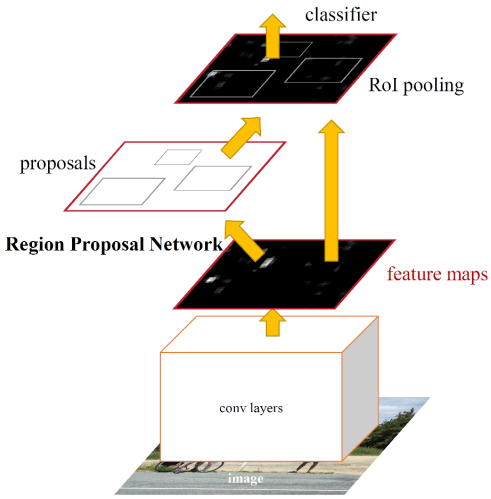


Figure 2.10: The Faster RCNN pipeline. The figure is from [8].

tions to the training process, reduced the time required for inference and training, as well as leading to higher prediction accuracy.

In 2016, Faster RCNN further improved upon the Fast RCNN and RCNN architectures. It introduced a Region Proposal Network (RPN) to perform the region proposal step[8]. In the two earlier methods this had been done through a method called Selective Search, which is a hierarchical segmentation method which repeatedly merges regions based on a similarity measurement[51]. This is a reasonable fast method, but became the bottleneck in the Fast RCNN architecture. Faster RCNN solves this by training a separate model, the RPN, to propose ROIs. This RPN works on a feature map produced by the feature extractor. This further improves both execution time and performance.

Faster RCNN's modular architecture has facilitated easy modifications. Mask RCNN is one such modification. It appends a new branch for predicting a detected object's mask in parallel with the existing branches for bounding box regression and class prediction[9]. This makes Mask RCNN an architecture for instance segmentation, as discussed in Section 2.1. This thesis uses the Mask RCNN architecture for its analysis. Mask RCNN also introduced one more modification to the Faster RCNN architecture, the RoIAlign pooling method. It differs from RoIPooling in that it replaces the quantization step with a bilinear interpretation step. This is needed because the original RoIPooling method led to pixelwise misalignements in the ROI and the extracted feature maps[9].

Since the mask prediction is an image-to-image operation, the creators of Mask RCNN used a Fully Convolutional Network (FCN) to perform this task. A FCN is a type of CNN that consists only of spatially bounded operations like convolutions, pooling, and the transpose convolution. FCNs usually perform semantic segmentation, but since it works within a ROI it performs instance segmentation in the Mask RCNN architecture. Fully Convolutional Network (FCN) will be further discussed in Section 2.3.7.

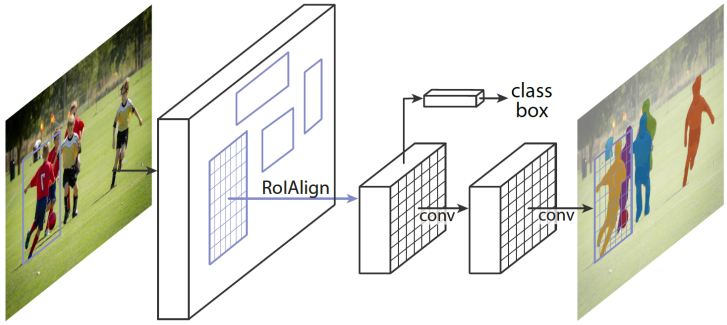


Figure 2.11: The Mask RCNN architecture. The figure is from the Mask RCNN paper[9].

2.3.6 ResNet

One defining aspect of Deep Learning is that the networks gradually extracts information with higher and higher semantic value further into the network structure. This should allow the network to make decisions and calculations based upon high semantic information and thus achieve high performance. In theory, this should make deeper networks, networks with more layers of artificial neurons, better than shallow ones. There is only one problem: in many applications deeper networks do not perform better than shallow networks, often they even perform worse. When increasing the number of layers within a neural network, the prediction accuracy first levels off, before it decreases rapidly as the number of layers further increases[10]. One might imagine that this reduced accuracy is caused by overfitting due to the increased number of parameters in the deeper network. However, experiments performed in [10] found this to not be the case, since the performance on the training data also decreased. Overfitting is discussed in Section 2.3.11.

There have been made several attempts to mitigate this problem, and one of them is incorporated in the network architecture ResNet. It was presented by a Microsoft research group in 2015 and won first place in the ILSVRC2015 (ImageNet Large Scale Visual Recognition Challenge) competition. The researchers hypothesised that the reduced accuracy of deeper networks was caused by the network's inability to learn identity transformations.[2] This fact, combined with the difficulty of optimizing the parameters of the network layers, means that some network layers actually reduce the semantic value of the feature maps, instead of increasing it. To mitigate this, the researchers reformulated the network layers to learn residual functions with respect to the input instead of learning unreferenced functions[10]. This was done to make it easier for the network to learn identity transformations, an ability which was theorized to improve performance. Their theory was strengthened as their modification led to increased performance. One such residual layer is shown in Figure 2.12.

[2]Identity transformations are transformations that copy the input to the output.

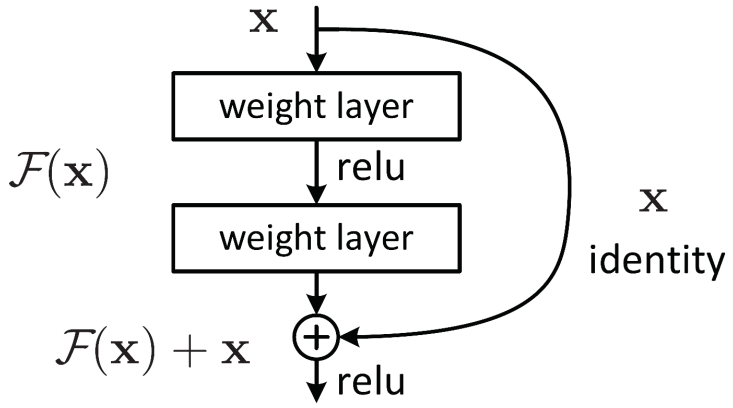


Figure 2.12: The residual layer used in the ResNet architecture. The figure is from [10].

2.3.7 Fully Convolutional Neural Networks

A Fully Convolutional Network (FCN) is a network architecture that does not use FC layers. This means that it can operate on inputs of various sizes and perform tasks on a pixel to pixel basis. This section discusses the components often used in this kind of network architecture and some of its variations.

The shape of neural networks often depends on what tasks they do. Networks that classify images shrink as the input propagates through its layers, as it gradually extracts information of higher and higher semantic value from the input. This tendency is furthered by the implementation of convolutional layers and pooling layers, which generally reduce the size of the input. When researchers at the Computer Science department at Berkeley wrestled with the task of creating a system that could perform semantic segmentation, that means classifying each pixel in an image with a class, they were faced with the opposite problem, they had to increase the size of the feature maps[11]. This was required because the semantic segmentation task requires that there is a one-to-one correspondence between the input pixels and the output pixels in the network. This problem also arises in other network types, such as Generative Adverserial Networks (GANs), which are also often required to increase the size of the feature maps.

The operation that performs the upsampling operation has many names, but transpose convolution seems to be preferred in the literature. Other popular names include: deconvolution, upconvolution, fractionally strided convolution, and backward strided convolution. The transpose convolution operation can be implemented in multiple ways: as a convolution with specially designed zero padding emulating a fractional stride length or by, after reformulating the convolutional operation as a matrix multiplication, using the transpose of this matrix to find its input which has a larger size than the output.

In 2015 the researchers from Berkeley discovered that the performance of the semantic segmentation could be drastically improved by concatenating feature maps from earlier layers with the

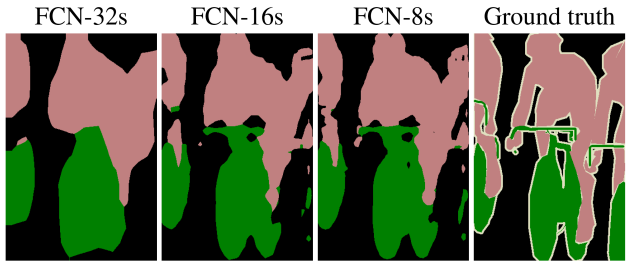


Figure 2.13: Different semantic segmentation performances achieved by researchers at Berkeley[11]. The pink color represents pixels classified as cyclist, the green pixels have been classified as bike-pixels, and black pixels have been classified as background.

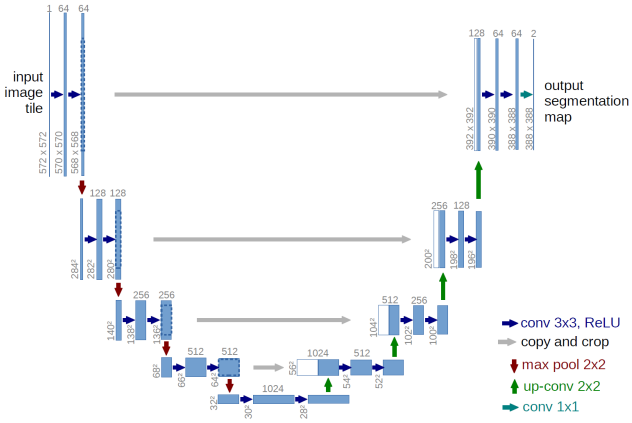


Figure 2.14: The U-Net fully convolutional network structure. The figure is from the paper debuting the U-net architecture[12].

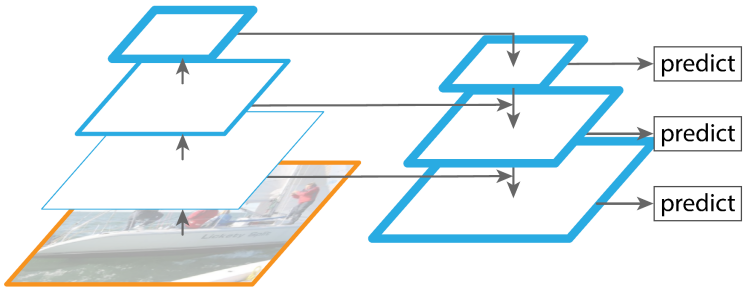


Figure 2.15: The FPN. The figure is from [13].

outputs from the transpose convolutional layers. This allowed the network to combine the high resolution spatial information from the early layers with the highly semantic information from the upsampled layers, resulting in a more high resolution semantic mapping. Figure 2.13 shows the different semantic segmentation performances achieved by fusing different feature maps at different depths with the output of the transpose convolutional layer. The leftmost figure is the performance achieved without any fusion, the left-center figure fuses an early feature map with the output, the right-center figure additionally fuses an even earlier layer, the right figure is the ground truth.

The technique of combining high resolution feature maps from early layers with upsampled feature maps in later networks has led to several different network architectures. One of these is the U-Net architecture, which was made to perform biomedical image segmentation[12]. It is shown in Figure 2.14. Here the connections between earlier layers and the upsampled layers later in the network are clearly visible. Fusing high resolution feature maps containing low semantic information with low resolution feature maps containing high semantic information is also used in FPNs, which are described in Section 2.3.8. The mask regression head in the Mask RCNN used in this work is implemented as a six layer FCN, consisting of 5 convolutional layers and 1 transpose convolutional layers, without any skip layers.

2.3.8 Feature Pyramid Network

As mentioned in Section 2.3.7, convolutional neural networks have an inherit trade-off. As the inputs propagates through the networks the information's semantic value increases, but the spatial information is degraded due to the reduced resolution of the feature maps. In 2017 the Facebook AI Research group published a new network architecture that does not suffer from this effect[13]. This architecture is called a FPN and consists of two main components.

The first component is a regular convolutional network, which suffers from the trade-off discussed above. A secondary, upside-down convolutional network is then introduced in parallel to the first convolutional network. This secondary network uses transpose convolutions, which were discussed in the previous section, to increase the size of the feature maps. Layers in the networks with the same spatial size are connected by lateral connections, which allows spatial information from

the early layers of the first network to be combined with highly semantic information upsampled through the second network. In their paper the Facebook AI researchers specify the lateral connections as 1x1 convolutional layers, which allows the network to learn which features from the first network are beneficial to transfer to the second network, as well as convert it to the correct channel depth. The FPN architecture is illustrated in Figure 2.15.

The FPN architecture is able to create highly semantic, high resolution feature maps, which naturally allows higher performance. In their paper from 2017 the researchers from Facebook AI research group demonstrated state-of-the-art performance on the COCO object detection benchmark by modifying a Faster RCNN to use the FPN architecture. A FPN is also a central part of the MaskRCNN implementation used in this thesis and is used to improve the performance of the feature extraction backbone at little extra computational cost[2].

There is one detail regarding the FPN architecture left to discuss. That is which of the feature maps generated by the feature pyramid should be used in the classifier head, bounding box regression head, and mask regression head of the Mask RCNN architecture. As a reminder: in the Mask RCNN architecture the Region Proposal Network (RPN) scans the feature maps looking for regions of interest. These regions of interest are used to generate cropped versions of the feature maps using the RoIAlign algorithm. The researchers at Facebook AI research group proposed to use

$$k = \lfloor k_0 + log_2(A_{roi}/A_{im}) \rfloor \tag{2.15}$$

to determine which feature map to use. In Equation 2.15 the level of the feature pyramid to use is represented by k, k_0 represents the target feature map if the region of interest covers the entire image, A_{roi} is the area of the region of interest, and A_{im} is the area of the image.

2.3.9 Training

The behaviour of a machine learning system is decided by the values of its learned parameters. In a Deep Learning based system, for instance a CNN classifier, these parameters are the weights and biases in the kernels and fully connected layers. Training a neural network is an optimization problem whose dimensions are the network's weights and biases and whose loss function is defined by the programmer. This formulation allows the use of a wide array of tools and algorithms for optimization. The complexity of the problem restricts the use of closed form optimization techniques. Second order optimization techniques, optimization techniques that incorporate estimates of the curvature of the optimization process, have seen limited use due to their large storage space and computational requirements. This makes first order optimization techniques the most popular for optimizing neural networks, even though they are less effective per optimization step and often require extensive hyperparameter tuning[52].

Currently, the most used method for training neural networks is called gradient descent. Three optimization paths generated by gradient descent is illustrated in Figure 2.16. Gradient descent minimizes the loss function by calculating the gradient of the loss function with respect to the parameters of the network. These are then changed in the opposite direction of the gradient. Over several iterations this minimizes the loss function, and over time the loss function's value converges

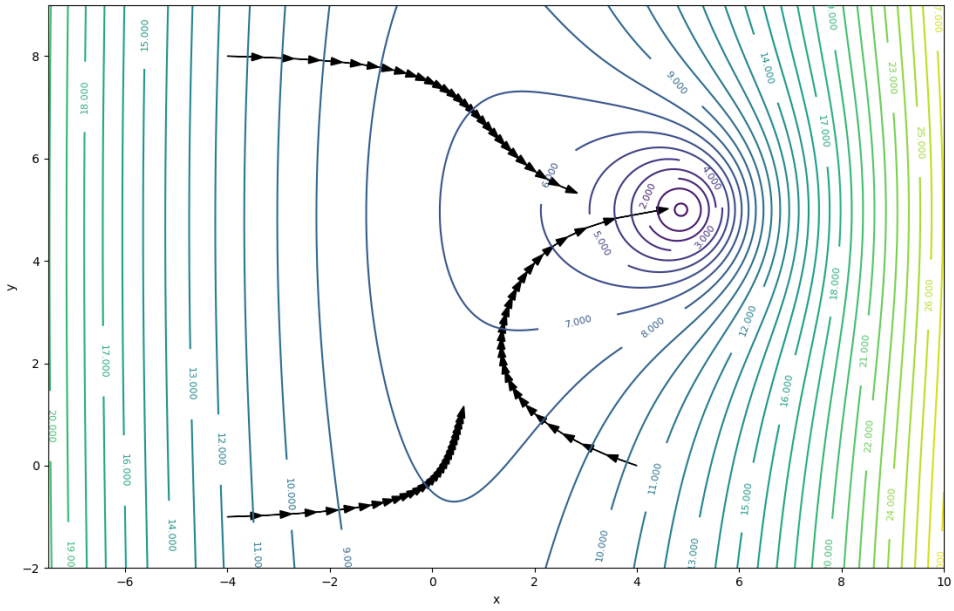


Figure 2.16: Three different gradient descent optimization paths with different initial conditions. The function being minimized is $L(x, y) = 10 - \frac{15}{\sqrt{(x-5)^2+(y-5)^2+1}} + 0.2x^2 + 0.1sin(0.05y)$, the learning rate is 0.4, and the algorithms have performed 30 optimization steps. Note how the paths follow the loss function's gradient even though it isn't the most efficient path. The figure is created with a Python script and the Matplotlib module.

to an optimum value. Note that this value might not be the global optimum value, but rather a local optimum. The mathematical discussion in this section is based around a traditional fully connected artificial neural network. The gradient descent update rule for a weight between two neurons in a fully connected artificial network is written as

$$w_{ij}^{n} = w_{ij}^{n} - \eta \frac{\partial C}{\partial w_{ij}^{n}} \tag{2.16}$$

where w_{ij}^{n} is the weight between neuron i in layer $n-1$ and neuron j in layer n. η is the learning rate, which decides how much the weights are nudged. $\frac{\partial C}{\partial w_{ij}^{n}}$ is the partial derivative of the loss function with respect to w_{ij}^{n}. The update rule for the bias term is similar, with b_{i}^{n} replacing w_{ij}^{n}. Equation 2.16 only updates one weight, when a model is trained this function is usually applied to all weights and biases in the network many times.

A classical implementation of gradient descent would minimize the average loss function of all the training samples. However, in many cases involving large datasets, this approach is prohibitively slow. For example: if a machine learning project has a dataset of 10000 images it would need to average all the losses of all the images before performing a single training step. Stochastic gradient descent[53] and mini-batch gradient descent was introduced to speed up this process. In mini-batch gradient descent the loss value is averaged over a subset of data samples, and in stochastic gradient descent the training step is performed with regards to only a single data sample. Naturally this speeds things up. Continuing on the previous example, the stochastic gradient descent would, in theory, perform 10000 training steps for each training step the classic gradient descent performed. This drastic speed up comes at the cost of more noisy training progression and potentially lower end performance, since the effect of single samples are not smoothed by the averaging process as it is in gradient descent. Mini-batch stochastic gradient descent represents a compromise between these two extremes, and presents the batch size as a tunable hyperparameter. A training step using mini-batch stochastic gradient descent can in some cases be as quick as a training step using stochastic gradient descent thanks to the parallelism of modern computers allowing all of the samples in the mini-batch to be calculated in parallel. In that case there are few reason for not using it.

There are several kinds of loss functions, each for different network applications. They are all related in that their job is to compare the predicted output of a machine learning system with the ground truth and describe the error through a scalar value. Typically a deciding factor for which loss function is used is whether the network performs regression or classification. In classification the output is often a probability distribution over the possible results and there are loss functions, such as cross-entropy loss, specially designed to take advantage of this. If the network performs regression the output is typically a vector or a scalar, and again there are loss functions, such as L2 loss or Huber loss, designed for this purpose. The Mask RCNN implementation used in this work uses cross-entropy loss to evaluate the classification and mask prediction, and Huber loss to evaluate the bounding box prediction performance.

As seen in Equation 2.16 the change of a network parameter is proportional to the gradient of the loss function with respect to that parameter. This is intuitive, the gradient of the loss function represents the direction of most rapid increase of the loss value. If the parameter is changed in the opposite direction, the loss function value will decrease. A method for finding these gradients for all

the network parameters is called backpropagation. When performing inference, which means using the model to calculate some prediction, the input flows from the input layers towards the output layers, through the intermediate layers. This produces a result, and an associated loss defined by the loss function. Backpropagation, as the name implies, works by propagating the gradient of the loss function backwards through the network from the output layer towards the input layer. This is done by repeated execution of the chain rule. Remember that the chain rule implies that

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

if $z = f(y)$ and $y = g(x)$. Backpropagation uses this to find $\frac{\partial C}{\partial w_{n-1}}$ based on $\frac{\partial C}{\partial w_n}$ by propagating the gradient through the activation function and linear transformation of the $(n-1)$th layer by applying the chain rule. To be clear: n is the layer number, C is the loss function, and w is the vector of layer weights. In other words: it uses the gradients found at one layer to find the gradients in the previous layer. This backpropagation of gradients is the defining feature of backpropagation. This thesis will not go into more depth than this, for more information the reader is referred to Chapter 2 in [43] for a more comprehensive description.

The inefficiencies of gradient descent have sparked the use of several modifications intended to improve its performance. One of these modifications is called momentum and lets the neural network learn quicker if the calculated gradients are similar over several optimization steps. As implied by its name, it works by giving the parameters being optimized a momentum. It does this by keeping track of the changes to each parameter through a variable which is updates by

$$\Delta \mathbf{w}_t^n = p\Delta \mathbf{w}_{t-1}^n - \eta \nabla_{\mathbf{w}} C \tag{2.17}$$

where $p \in [0, 1]$ is a momentum parameter, $\Delta \mathbf{w}_t^n$ is the change to the nth weight matrix in the network at optimization step t, η is the learning rate, $\nabla_{\mathbf{w}} C$ is the gradient of the loss function with respect to the weight matrix. In the two dimensional case this update rule emulates the movement of a physical object on a surface, whose height is defined by the loss function. A steep slope, as defined by the gradient of the loss surface, gives the object a velocity. This velocity then causes the position of the object, which represents the parameters of the network, to change. This change brings the position of the object to a new point on the surface, leading to a new change in the velocity caused by the gradient of the loss function at the new point. This effect continues until the parameters have converged to a local optimum. The momentum parameter also emulates a friction effect. A momentum parameter close to 1 emulates a weak friction effect, allowing the parameters keep changing even though the gradient is no longer steep, while a momentum parameter close to 0 emulates a strong friction effect, quickly stopping the parameters from changing if the gradient disappears. The Mask RCNN implementation used in this thesis uses momentum in its training procedure. The mathematical description regarding the momentum technique is from [54], with some slight modifications.

2.3.10 Batch Normalization

Batch normalization is a technique used extensively in the Mask RCNN implementation experimented on in this work[2]. It is an extension of input normalization, which is commonly applied

in machine learning. Various input normalization methods have been proposed and the techniques have been shown to improve network performance[55] and accelerate the training process. A simple example to motivate how input normalization can speed up the training process is to recall how the sigmoid activation function suffers from gradient saturation when the input to the neuron is particularly large or small. If, as an example, a neural network was to perform some classification task on vectors of housing data the neurons would saturate quickly if the house-price was expressed in dollars, since the large values would cause the neurons to suffer from the saturated gradient effect.

The batch normalization technique was debuted in a paper released by Google in 2015[56]. It works by normalizing the input activations to the layers in neural networks by subtracting the mean and dividing by the variance of the activations. To motivate this they define the term *internal covariance shift*, which is an extension of the term *covariance shift*. *Covariance shift* occurs when the input distribution to a learning system changes. For example: if a machine learning classifier has only been trained to detect green apples and is then asked to detect a red apple, the system can not be expected to perform very well since it would suffer from *covariance shift*. *Internal covariance shift* expands on this by claiming that this effect also occurs during training of neural networks. It goes like this: during the training process, the parameters of the network changes. This changes the output activation distributions between the layers and inhibits the learning of the intermediate layers since their input distributions change. This then leads to slower learning overall.

The researchers at Google found that normalizing each layer's inputs was computationally costly and not differentiable everywhere. They made two simplifications to alleviate these problems. The first simplification was to normalize each single activation individually instead of the entire activation vector. For a layer with d-dimensional input $x = (x^{(1)}, ..., x^{(d)})$ they normalized each dimension according to

$$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{Var[x^{(k)}]}} \tag{2.18}$$

where the expectation and variance are computed over the training data set[56]. Now over to the second simplification. Instead of calculating the expectation and variance over the entire training data set, they instead estimate them using the samples in the mini-batches. For certain activation functions, such as the sigmoid function, this normalization could inhibit the layer's ability to access its non-linear sections. To combat this, the batch normalization technique adds a learnable linear transformation to the output of each normalization. The linear transformation is formulated as

$$y^{(k)} = \gamma^{(k)}\hat{x}^{(k)} + \beta^{(k)} \tag{2.19}$$

where $\gamma^{(k)}$ and $\beta^{(k)}$ are the learnable parameters of the linear transformation. By learning this transformation the batch normalization is able to learn the identity transformation, if proves to be beneficial with regards to the loss function[56]. During backpropagation the network learns the optimal $\gamma^{(k)}$ and $\beta^{(k)}$ along with the weights and biases. In many cases the network does not have access to a batch during inference. This would make it impossible to estimate $E[x^{(k)}]$ and $Var[x^{(k)}]$. To combat this the implementations of batch normalization usually keep track of these two estimates with a rolling average filter and uses the latest value from training when performing inference[56].

There are a few more things to note about batch normalization. Firstly, the discussion above has been with regards to implementing batch normalization between fully connected layers. In the Mask RCNN implementation the batch normalization is between convolutional layers, which means the batch normalization requires some modifications. However, the intuition is the same, so the curious reader is referred to Section 3.2 in [56]. Secondly, newer works such as [57] have suggested that the hypothesised reason for batch normalization's effectiveness is incorrect, and that it instead smoothes out the optimization landscape[57]. It is however certain that batch normalization accelerates the network's training process, which is why it is still being used, even though its exact functioning is under debate.

2.3.11 Overfitting and Generalization

A problem plaguing not just Deep Learning models but also machine learning models in general, is overfitting. Overfitting occurs when the model memorizes the training dataset instead of learning its general features. This can be difficult to detect since the performance of the model will continue to increase as normally when tested on the training dataset even though the network's practical performance actually worsens. This is analogous to a student that just memorizes all the practice assignments instead of actually learning the topics. His or hers performance on the practice assignments will steadily increase, but the person will not actually learn anything useful that can be applied to solve new tasks. In Deep Learning terms, the student would have overfitted on the training assignments and failed to generalize the knowledge.

A model's tendency to overfit on the training data increases with a decreasing dataset size. One reason for this is that the model might pick up on patterns present in the small dataset which do not exist in the real world, but instead caused by sampling noise. This false pattern might be attenuated if the size of the training dataset is increased. Another factor is the complexity of the model. In general, more complex models overfit more readily. This might be because a simpler model is forced to use the most general patterns in its predictions, while a complex model can afford to base its predictions on minute patterns which might be caused by noise.

Because of their large number of parameters, Deep Learning based methods are very susceptible to overfitting. Virtually all Deep Learning models will overfit on the training data if given the chance. Some modern Deep Learning models, for instance the GPT-3 Natural Language Processing (NLP) model, can have over 100 billion parameters[58]. As discussed in previous sections, a parameter can be a weight between neurons, a bias, or a weight in a kernel. This huge parameter space makes simply memorizing the dataset a valid strategy. As expected, this strategy will not perform well when tested on new data and is therefore undesirable.

There are several methods to combat overfitting. The most common is to use part of the training dataset as a validation dataset, sometimes called the hold-out dataset. The model's performance on the validation dataset is periodically measured, and since the model is not allowed to train on the validation dataset it is an unbiased measure of the model's performance, as long as the validation dataset actually represents the real world data the model will encounter in a realistic manner. The validation dataset is normally 5% - 30% of the training dataset. A larger validation dataset leads to stronger protection against overfitting, but also a smaller training dataset, since the validation dataset

cannot be used in the training process to prevent the model from overfitting on it. Another popular technique against overfitting is called early stopping. This technique simply stops the training process if the model's performance on the hold-out dataset stops improving. In practise the model's performance is often required to decrease over several training steps before the training is stopped.

A general intuition regarding overfitting in machine learning systems is that an overfitted system has learned patterns from the training dataset that is generally not present in real world data. This intuition forms the basis of several methods to combat overfitting by forcing the models to learn strong patterns, with the idea that these are more likely to be encountered in real world data. L1 and L2 parameter regularization does this by including a penalty term in the loss function which penalizes the L1 norm or the L2 norm of the network parameters respectively. Both of these regularization methods apply a decay to the network weights, forcing the neurons to activate on strong patters[43]. The Mask RCNN implementation used in this work uses L2 regularization in its training procedure.

Another method that works by the same intuition is the dropout regularization method. When applying this method in the training procedure, for each training step, a subset of the network layers' activations are randomly set to zero. There are two intuitions motivating this. Firstly, this process adds noise to the learning process which forces the neurons to learn strong features that can overcome this noise[59]. Secondly, the dropout method forces the network to apply a form of ensemble learning since the dropout splits the network into several sub-networks[59]. This further forces the network to rely on strong features in its predictions.

It is worth noting that batch normalization adds a slight regularization effect to the network in addition to its beneficial effect on training speed. According to the paper on batch normalization[56], it adds a non-deterministic nature to the network which was beneficial in their experiments. The final anti-overfitting method mentioned in this section is to expand the training dataset via dataset augmentation. This is done through the creation of new samples by altering some original samples through image transformations. Transformations that are often used include: flipping the image, rotating the image, cropping the image, applying noise, warping the image, swapping the color channels, etc. More recent methods also include training GANs to perform dataset augmentation through style transfer[60][61]. Since dataset augmentation increases the size of the training dataset and a network's tendency to overfit is reduced with an increased training dataset, it is clear that the method could reduce a model's tendency to overfit. Since this work uses entirely synthetic training datasets to train the Mask RCNN system the size of the datasets are not a concern. However, whether the synthetic maritime datasets accurately represent images of real world maritime scenes is a relevant question. That question falls under the topic of domain adaption and will be discussed in Section 2.4.1.

2.3.12 Transfer Learning

As described in previous sections, training Deep Learning based methods require large amounts of both time and data. Increasing the size of the dataset reduces the model's tendency to overfit and increases performance, while sufficient training time is required for the model's performance to converge. This large demand for training data and training time often restricts smaller experiments like the ones performed in this thesis due to lacking the resources required to collect the dataset as well as

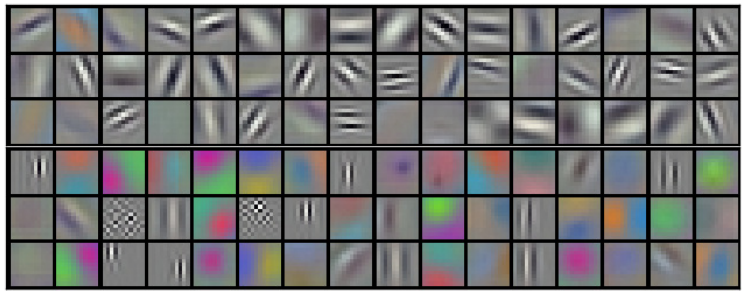


Figure 2.17: 96 filters from the first layer of a convolutional network trained to classify images. The figure is from [14].

the computational requirements involved in the training process. To illustrate this: the initial Mask RCNN implementation by Facebook AI Research group took 44 hours to train when it was configured with a ResNet-101-FPN feature extraction backbone and trained on 8 high performance GPUs with an effective mini-batch size of 16 images[9]. The Facebook AI Research group highlights this as a relatively quick training time, but still, most smaller projects such as this one do not have access to the same level of computational power.

Fortunately there are methods that reduce both the amount of training data and training time required for Deep Learning systems. This thesis will employ a method commonly used for this, called transfer learning. Transfer learning is based upon transferring the knowledge a Deep Learning system learns when trained to perform one task, to a new, but related task. The practical details on how this is achieved is dependent on the network architecture being modified. As noted in 2.3.3, CNNs classifiers often consists of two parts: the feature extractor, which consists of convolutional layers, and the classifier, which is often implemented with FC layers. When performing transfer learning on a CNN classifier it is common to copy the network parameters from the initial trained network, freeze the parameters in the feature extractor and only train the classifier part of the network. Naturally, the dimensions of the network layers might not be compatible. This might happen if the number of output classes or the input dimensions in the new task is different is different from the original task.

Using transfer learning can drastically reduce the required data and time in the learning process. The project thesis preceding this thesis[1], showed that when using transfer learning, the training time required for the network's performance to converge was reduced to ≈ 10.5 hours when the network was trained on a single high performance GPU with an effective batch size of 2 images and configured to use a ResNet-101-FPN feature extraction backbone. That is a massive decrease in training time, especially when considering that much less computational power was available. In that application all the network's layers, except the final prediction layers, were pretrained on the COCO dataset which contains ≈ 200000 labeled images containing 80 different classes. The final prediction layers were not copied over from the original network since the number of predicted classes was different in the new task. The networks in this thesis have been pretrained on the COCO dataset, and thus take advantage of the benefits of transfer learning.

	Ground truth		
Predicted Result		True	False
	True	True positive	False positive
	False	False negative	True negative

Table 2.1: The relationship between the ground truth and predicted results.

So why does transfer learning work so well, even if the new task is quite different from the original task? The basic intuition is that copying pretrained parameters into a new network gives it a head start, since many of the pretrained sub-tasks performed inside the network are universally useful. Think of how the early layers of a convolutional network, some examples are shown in Figure 2.17, search for very general image features. These general features can be horizontal or vertical edges, a specific color, etc. When training the new network using the pretrained parameters it is likely more efficient to learn how to reuse these existing filters than to learn them from scratch. In cases where only a small dataset describing the new task is available, transfer learning could in theory improve the final performance of the network since the reduced training time could reduce the network's tendency to overfit on the limited dataset.

2.3.13 Performance Measurements

To facilitate accurate comparisons between different computer vision systems it is important to have precise measurements of a system's performance. There exists many different measurements, based on which aspect of the system's performance is being evaluated. The Mask RCNN architecture performs four main functions: it detects objects, classifies them according to the defined classes, predicts a bounding box, and creates a segmentation mask covering the detected objects' silhouettes. Because of the system's multiple outputs several different performance measures are required to measure its entire performance.

The performance measures for object detection will be considered first. Measures describing the performance on this task should describe how accurately the system detects objects that actually exist in the image and then classifies them correctly. The measurements should also describe how often the model sees objects that are not actually present in the image. A measurement framework based upon true positives (tp), false positives (fp), true negatives (tn), and false negatives (fn) is often used. This framework is most easily illustrated with a table. Table 2.1 shows the relationships between the ground truth and the object detection system's output. An object detection system performs well if it exhibits few false negatives and false positives. Once calculated, these measurements are often combined into the compound measures called precision and recall.

$$precision = \frac{tp}{tp + fp} \tag{2.20}$$

$$recall = \frac{tp}{tp + fn} \tag{2.21}$$

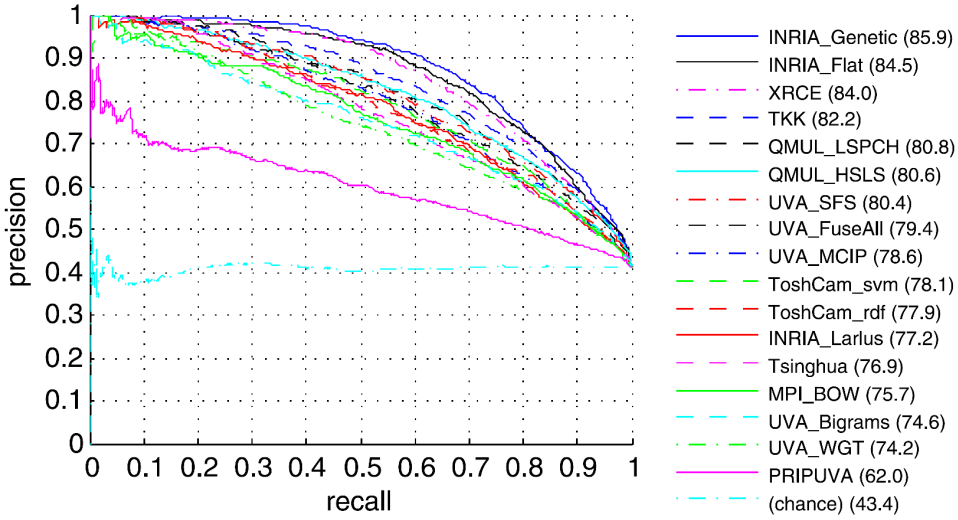


Figure 2.18: Precision-recall curves for different object detection algorithms. The figure is from [15].

The precision measure indicates how accurate the object detection system is by calculating the quotient of all correct positive detections and all positive detections. A high precision value indicates that if an object has been detected, it likely is actually present in the image. Recall, on the other hand, describes how accurately the system detects all existing objects. This is done by calculating the quotient of all correct positive classifications and all existing positive objects. A high recall value indicates that if there exists an object in the image, the detection system will likely detect it. Equation 2.20 and 2.21 shows how to calculate these values. A high performance object detection system should have precision and recall values close to one. That would indicate that the system correctly detects all objects that actually exists in the input image without 'hallucinating' objects.

These two measurements are commonly plotted in precision-recall plots. The plots are generated by collecting a sample of predictions generated by the object detection system, sorting them with regards to their predicted accuracy, and then iterating through them while calculating the cumulative precision and recall. These cumulative values are then plotted against each other. Examining a precision-recall plot provides insight into how precise the model is at various sensitivity levels. Figure 2.18 illustrates a handful precision-recall curves. The information of a precision-recall curve can be further condensed into an Average Precision (AP) value. The AP value is defined as the area under the precision-recall curve. It is calculated by

$$AP = \sum_{r=0}^{1} (r_{n+1} - r_n) p_{interp}(r_{n+1}) \tag{2.22}$$

$$p_{interp}(r_{n+1}) = \max_{\tilde{r}:\tilde{r} \geq r_{n+1}} p(\tilde{r}) \tag{2.23}$$

where AP is the average precision value, r_n is recall value number n, and $p_{interp}(r)$ is the precision value at a given recall value. AP is calculated using the interpolated precision value shown

Figure 2.19: Examples of various bounding box predictions and their corresponding IOU scores. The figure is from [16].

in Equation 2.23 to combat the 'zig-zag' nature of the precision-recall curve caused by the way the plots are generated. Depending on which benchmark the metrics are computed for, there might be an additional metric called mAP in which the AP values corresponding to the different classes are averaged together. However, most modern object detection systems report their performance using the Common Objects in Context (COCO) benchmark, which does not separate AP and mAP, and report all AP values after they have been averaged over all the classes.

When measuring the performance of classifier networks with multiple defined classes a confusion matrix can be a helpful tool. A confusion matrix provides insight into how often the network mistakes one class for another class in its predictions. This is achieved by creating a matrix in which the ground truth classes are represented in the columns and the predicted classes in the rows. Then, for each ground truth class the probability distribution over the predicted classes is filled in. One way to use a confusion matrix is to first look up a ground truth class and then look at the class probability distribution produced by the network when trying to predict it. If the network performs well the probability of all other classes than the ground truth class should be very low.

To measure the performance of the bounding box and mask prediction, IOU is commonly used. IOU measures how accurate a predicted bounding box or a segmentation mask is relative to the ground truth. A few bounding box predictions along with their corresponding IOU scores are illustrated in Figure 2.19. IOU can be formulated as

$$IOU = \frac{BB \cap GT}{BB \cup GT} \tag{2.24}$$

with BB being the area covered by the bounding box (or segmentation mask) and GT being the area covered by the ground truth. A good prediction IOU should be close to 1, while a prediction that completely misses has an IOU equal 0. Which IOU value is used as the threshold for deciding whether a detection is correct depends on which benchmark is used. Usually an IOU above 0.5 is required. The COCO challenge, a popular benchmark for instance segmentation, tests the systems using ten different IOU thresholds[62].

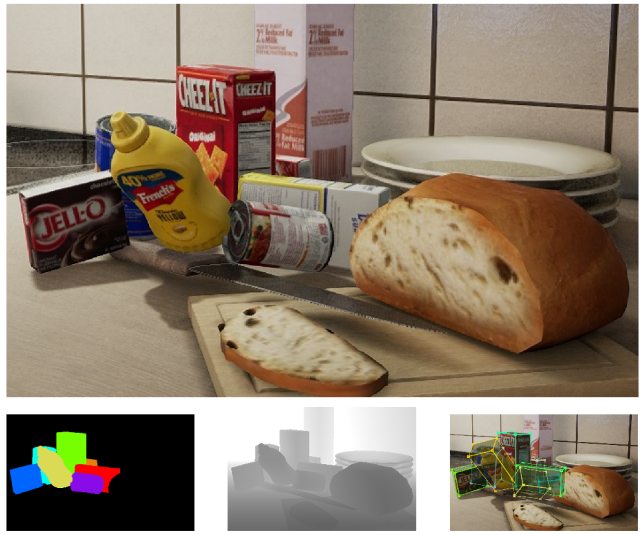


Figure 2.20: A synthetic image with class labels, depth-map, and 3D bounding boxes. The figure is from [17].

2.4 Synthetic Dataset Generation

Since the performance of many machine learning systems is limited by the availability of training data, synthetic dataset generation could by first glance seem as the key to it all. This is probably why it has been applied so often and in so many machine learning areas. Take instance segmentation as an example: imagine the task of creating a dataset containing ships in various maritime scenes. Collecting the images is easy enough; one could either venture out with a camera, or scrape the internet for images. However, before the dataset is ready for use in supervised machine learning it must be labelled. What is involved in the labelling process depends on what task the machine learning application is going to perform. In the case of instance segmentation, it means manually detecting, classifying, and masking all the objects of interest, so the system has something to mimic while training. This can be very time consuming. As a comparison, the system used in this thesis can create several thousand perfectly labelled training samples overnight. The allure of synthetic dataset generation is obvious.

Naturally, synthetic dataset generation also requires quite a lot of labour. To illustrate: this thesis uses a 3D graphics rendering program, 150 highly detailed 3D models, as well as the control program for the dataset generation. All of these components require large amount of skilled work, unlike manual dataset collecting and labelling, which can be performed by almost anyone. But once these components have been made they can create a dataset of desired size and content, as long as one has 3D models representing the desired objects. The 3D graphics rendering program can also be used for other projects within the business, as is the case with the Kongsberg Cogs graphics engine used in this thesis. If the dataset generation is efficient enough it is also possible to generate the dataset on the fly, removing the large data storage requirements often associated with machine learning. Another useful feature of synthetic dataset generation is the ability to extract information

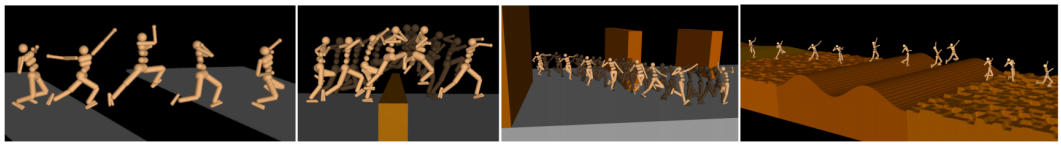


Figure 2.21: A reinforcement learning agent that has learned to walk in a virtual environment. The figure is from [18].

from the simulated scenes that would be too difficult or costly to gather from real world datasets. Figure 2.20 illustrates this as the image includes a pixel-perfect depth-map and 3D bounding boxes. Both of which would be very difficult to collect from a real world dataset.

A different kind of synthetic dataset generation involves creating a completely virtual environment for the machine learning system to learn in. This is briefly mentioned in Section 2.2. This method does not create a dataset or generate one on the fly, but rather completely immerses the machine learning system in a virtual environment. This is commonly done when training reinforcement learning agents, since they learn by repeated trial and error. The time-acceleration offered by virtual environments allows the learning to be performed much quicker than if the agent was to learn in the real world, which seems to be locked at one second per second in most cases. This method of synthetic dataset generation is more difficult than creating a purely visual dataset since not only do the visual aspects of the simulation need to match the real world, but also the simulation's dynamics. One can imagine that a reinforcement learning agent trained to walk in virtual environment with less gravity or less friction would struggle to adapt to the real world without retraining. Figure 2.21 shows a series of frames from DeepMind's experiments on reinforcement agents learning to walk in virtual environments[18].

Synthetic dataset generation does not necessarily have to create the datasets entirely from scratch. In [63] the researches show that a simple synthetic dataset generation method based on cut and pasting object instances on to various real world backgrounds can be effective in training computer vision systems. This is a simpler method since it does not require the generation of synthetic 3D scenes or even 3D models, although it still requires some real world data collection in the background images and object instances. When applying this method it is difficult to extract more detailed information from the scenes, such as accurate depth-maps or 3D bounding boxes, which is easy when the entire 3D scene is simulated. It is unclear which method performs the best - it probably depends on the purpose of the machine learning program and the scope of the project.

Another important idea when generating a synthetic dataset as well as collecting a real world one, is to avoid any unintended biases in the data. For example: if an object detector is trained to detect oil tankers, but is only trained on images of red ships, the detector would likely be limited to detecting oil tankers of that color. As one can imagine, this would make the detector mostly useless in the real world. When using 3D models in the synthetic dataset generation this is avoided by to having a large selection of models for each class. If that is not possible, it a good idea to try to expand the available models by modifying the existing ones. One way to do this is to create additional models by altering the color or texture of some existing model.

2.4.1 Domain Adaption

A critical step in successfully applying synthetic dataset generation in a machine learning based computer vision product is the transition to the real world application. This involves adapting the machine learning system, which has only experienced synthetic images, for use with real world images. More rigorously: domain adaption means adapting a system that has only ever trained on $D_{syn} \sim p_{syn}$ to $D_{real} \sim p_{real}$, where D_{syn} and D_{real} is the synthetic and real world datasets and the distributions are represented by p_{syn} and p_{real}. There are several methods for making this transition as smooth as possible; a few will be discussed in this section.

The first method is the most intuitive one. It is to simply make the synthetic data distribution as similar as possible to the real world distribution. Or symbolically: $p_{syn} \approx p_{real}$. In the case of computer vision systems, this is achieved by using highly realistic 3D graphics with realistic lighting and other effects. The quality of the 3D models is also important, as well as the quality of any other visual objects in the scenes such as the ocean, the terrain, or the sky.

The next method is a little counterintuitive. It is called domain randomization and is based on showing the machine learning system a wide array of different samples during training, hoping that if the system can master the very varied training samples it can also master the real world dataset. Expressed in the framework of datasets and distributions it means making p_{syn} so wide that it encompasses p_{real}. This method is more pragmatic since it accepts that it is impossible to exactly match all real world cases with synthetic dataset samples and instead attempts to create a training dataset so diverse that it includes all the real world dataset as a subset. In [64] researchers from OpenAI implement a modified version of domain randomization to train a reinforcement learning based robot hand to solve Rubik's cubes. Their robot was trained in virtual environments which was gradually made more difficult and varied by their novel version of domain randomization called *active domain randomization*. Note that this version of domain randomization not only varied the visual aspects of the simulation but also the dynamics, since the reinforcement learning agent was taught both visual scene recognition and robot control.

2.4.2 Perlin Noise

In procedural terrain generation one of the main challenges is creating continuous, natural-looking topologies. The most common noise generation methods are not suitable for terrain generation since they create unrealistic, non-continuous terrain topologies. For example: a height-map where the height at each point is sampled from a uncorrelated uniform distribution will be full of high frequency variations in height, often called cliffs. Such a system might produce a height of 100m at ($x = x_0$, $y = y_0$), a height of 0m at ($x = x_0 + 1$, $y = y_0$), and a height of 50m at ($x = x_0 + 2$, $y = y_0$). Clearly this will not look realistic, normal terrain does not vary like this, at least not everywhere. To create natural looking terrain that mimics the effect of eroding forces such as wind and rain the terrain height at various points close to each other need to be correlated. In value based noise generators this is not the case, which leads to the unnatural looking jumps.

Gradient noise takes another approach to creating a noise-map. First, it creates a lower resolution noise-map of gradients, before recovering the high resolution height-map through interpolation. This

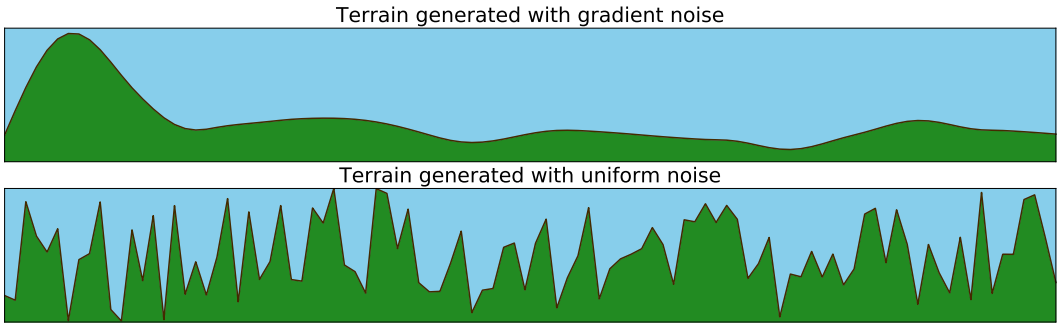


Figure 2.22: 1D procedural terrain generated with a gradient noise based method and with uniform noise. The figure was created with a Python script using the Numpy, Matplotlib, and Noise modules.

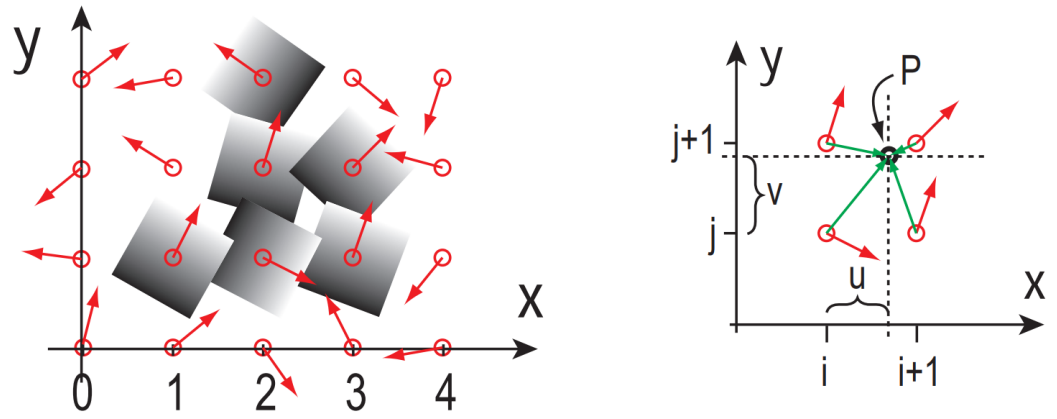


Figure 2.23: Figure illustrating the interpolation step in Perlin noise. It is meant as a supplement to the mathematics in this section. The figure is taken from Simplex Noise Demystified[19].

improves on value based noise since it forces the noise vary smoothly, which in turn generates more natural terrain topologies. Figure 2.22 shows how uniform noise is not suitable for creating realistic synthetic terrain topologies and highlights the difference between gradient based noise and value based noise, such as uniformly sampled noise. Perlin noise is one such gradient noise method. In the two dimensional case it works by generating a grid of psuedo-random 2D gradients. For a given point it then interpolates the value by using the four closest grid points. However, instead of using linear interpolation it uses

$$f(t) = 6t^2 - 15t^4 + 10t^3 \tag{2.25}$$

as the blending function. See Figure 2.23 for a visualization of the interpolation step. The following mathematical formulation is from Simplex Noise Demystified[19].

For point $P = (x, y)$, i and j is found as

$$i = floor(x)$$

$$j = floor(y)$$

this creates the distances

$$u = x - i$$

$$v = y - j$$

to the closest gradient grid points. The gradients at the four closest points are noted as g_{00} = gradient at (i,j), g_{10} = gradient at (i+1,j), g_{01} = gradient at (i,j+1), and g_{11} = gradient at (i+1,j+1).

The values derived at P from each gradient grid point is found as the dot products

$$n_{00} = g_{00} \cdot \binom{u}{v}$$
$$n_{10} = g_{10} \cdot \binom{u-1}{v}$$
$$n_{01} = g_{01} \cdot \binom{u}{v-1}$$
$$n_{11} = g_{11} \cdot \binom{u-1}{v-1}$$

These values are then blended together using the blending function shown in (2.25)

$$n_{x0} = n_{00}(1 - f(u)) + n_{10}f(u))$$

$$n_{x1} = n_{01}(1 - f(u)) + n_{11}f(u))$$

to finally create

$$n_{xy} = n_{x0}(1 - f(v)) + n_{x1}f(v))$$

which is the final value at the point P. Figure 2.23 illustrates the process. The left figure shows the low-resolution grid of 2D gradients and the right figure shows how the gradients are blended together to create the final height-map value. The program for synthetic dataset generation developed in this thesis uses Perlin noise when generating the terrain in the images. This enables the terrain to vary in the generated image samples, which in the end hopefully allows the trained instance segmentation algorithm to detect land in the input images.

2.4.3 Normal Mapping

As consumers demand higher fidelity 3D graphics, programmers continuously work to improve the performance of their graphics rendering programs with little extra computational cost. One way to achieve this in 3D graphics is called normal mapping. Normal-maps are used to improve the detail of 3D models or height-maps by encoding the surface normal vector at a point on the object $\boldsymbol{n}_{XYZ}$ in the RGB channels of an image. This enables the calculation of small lighting variations over the surface, without increasing the number of polygons in the 3D models, which would degrade the performance. This project used normal-maps to increase the visual detail level of the background

terrain in the generated dataset. The normal-map was generated from the height-map, and the following discussion highlights how it was done.

The normal vector can be found by first calculating the pixel-wise derivatives in the x and y direction. The height-map can be viewed as a function of the form

$$z = f(x, y) \tag{2.26}$$

where z is the height at point x,y. This can be rewritten to

$$F(x, y, z) = z - f(x, y) \tag{2.27}$$

from which, the gradient

$$\boldsymbol{n} = \nabla F(x, y, z) = (-\frac{\partial f}{\partial x}, -\frac{\partial f}{\partial y}, 1) \tag{2.28}$$

can be calculated. The terrain generation algorithm implemented in this thesis estimates $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$ by convolving the height-map pixel-values with the Sobel kernels

$$\boldsymbol{G_x} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{2.29}$$

and

$$\boldsymbol{G_y} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{2.30}$$

The gradient (2.28) is then L2 normalized

$$p(x, y)_{RGB} = \frac{\boldsymbol{n}}{\|\boldsymbol{n}\|} \tag{2.31}$$

and mapped to RGB values (0.0-1.0 for float and 0-255 for uint8) so the image can be saved as a .PNG file and imported into the Cogs graphics engine. The terrain generation system included in Cogs is discussed in Section 5.4.

2.4.4 HSV Colorspace

In computer graphics the color of a pixel is often represented in the RGB colorspace where value of each color channel determines the amount of that color present in the pixel. However, the RGB colorspace turned out to not be suitable for use in some aspects of the synthetic dataset generation. For example: to generate the color of some piece of background terrain one would have to sample three different probability density functions, one for each RGB value. It was not easy to determine realistic combinations of these colors when using the RGB colorspace. For example: how much blue should be present in the color of a rock? For most humans this is not an easy question. This

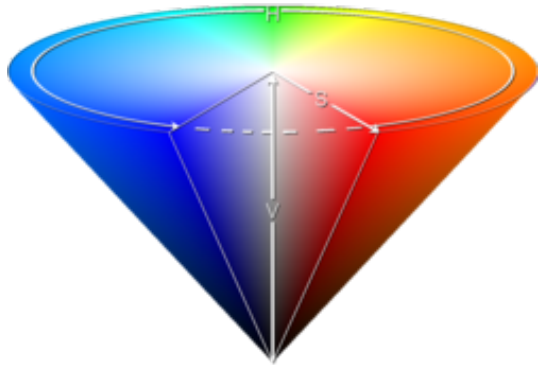


Figure 2.24: The HSV colorspace. The figure is from colorizer.org[20].

made the RGB probability distributions difficult to define, and therefore the RGB encoding was not suitable for most procedural generation applications.

The Hue Saturation Value (HSV) colorspace solves this by reducing the number of variables needed to describe the color's hue down to one. This is the hue value in the HSV colorspace. It also introduces variables describing the saturation and the brightness (value) of the color. The hue value is usually defined on the interval 0 - 360, while the saturation and brightness values are often defined on the interval 0 - 1. Since the hue of the color is a one dimensional value in the HSV colorspace it makes generating colors simpler and more elegant. For example: when defining the probability distributions for a piece of terrain one now defines the limits of the hue, the saturation, and the brightness of the terrain. Each with a separate intuitive influence on the resulting final color. Note that HSV and Hue Saturation Brighness (HSB) describe the same colorspace.

2.5 Sensor Fusion

Sensor fusion is a very complex topic and much can be said about it. However, this thesis will only cover the basic concepts needed to understand the motivation behind sensor fusion and the experiment discussed in Section 6.4 and 7.3.

As the name implies, sensor fusion is a method for combining information from different sensors with the goal of improving the performance of the unified system. The sensor systems often complement each other in some way, which makes the combined system more robust. One of the most common applications of sensor fusion is the combination of Inertial Measurement Unit (IMU) measurements with Global Navigation Satellite Systems (GNSS) measurements in order to improve the accuracy of a location service. These two sensor systems complement each other nicely. The Inertial Measurement Unit (IMU) measurements are highly accurate and provide high frequency measurements, but suffer from errors accumulating over time. The Global Navigation Satellite Sys-

tems (GNSS) measurements do not suffer from accumulating errors, but are much less accurate, less frequent, and might be unavailable if the GNSS antenna is prevented from receiving satellite signals. After sensor fusion however, the resulting location system supports accurate location measurements at high frequency with limited accumulated errors. The combined system can also operate for a limited time without satellite coverage. Fusion of IMU and GNSS data is certainly useful for autonomous vehicles, but as the focus of this thesis is on computer vision, it will focus on another application of sensor fusion.

The experiment discussed in Section 6.4 and 7.3 will examine how information from depth sensors can be combined with camera images to improve the performance of computer vision applications. For autonomous maritime vessels sensor systems such as Radio Detection and Ranging (RADAR) and Light Detection and Ranging (LIDAR) are most relevant. RADAR measure the time it takes for an emitted radar pulse to return to the transceiver and can provide the distances to detected objects. Its measurements can suffer from unwanted reflections from objects that are of no interest to the user, such as waves or foam. They are also often limited to detections in the horizontal plane. The positives are that the system can offer detection ranges of several tens of kilometers, depending on the geometry of the detected object, the RADAR wavelength emitted, the atmospheric conditions, the mounting configuration of the antenna, and much more. LIDAR also works by measuring the return time of an emitted electromagnetic signal. It uses higher frequency wavelengths such as ultraviolet light, visible light, or infrared light to produce its measurements, which is a 3D point cloud surrounding the sensor. A LIDAR sensor can produce very accurate point clouds, but is limited by its low range, typically between a couple of meters to several tens of meters.

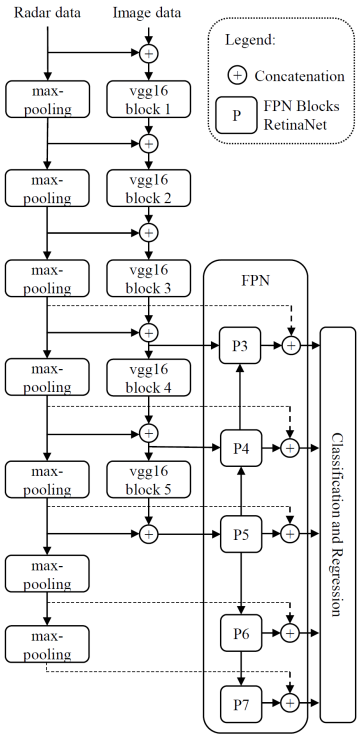


Figure 2.25: The method developed in [21] for gradually merging visual image data with radar data. The figure is from that paper.

Both these ranging sensors complement monocular camera systems in several ways. Firstly, both RADAR and LIDAR can operate in the absence of light, which severely degrades the performance of a visual camera system. They also provide depth information, which cannot be easily extracted from monocular imaging system without extra information. The experiment later in this thesis explores how to fuse depth information from such sensors with camera images in a neural network to achieve higher object detection performance. Primarily through a method inspired by [21], in which RADAR measurements were gradually fused internally in a neural network. Figure 2.25 shows the architecture the researchers employed to do this. They motivate their design by arguing that RADAR measurements are of higher semantic value than raw pixel data, so they should

be included in the network at a later stage of the feature extraction process. They admit that it is difficult to decide which layer of the networks the information should be included in, so their design allows the network to decide this on its own during the training process. For more information about this experiment, see Section 6.4 and 7.3.

Chapter 3

Estimating the Heading of Objects

This chapter discusses the topic of heading estimation. It first examines the definitions of heading and course, before it presents some modern sensor systems. It then continues on to visual heading estimation and what disturbances could impact such a system. Then, a few Deep Learning based approaches, both for the full 3D pose estimation task and the simplified in-plane orientation estimation task, are presented. Finally the three methods for heading estimation analysed in this thesis are discussed, along with the required modifications to the Mask RCNN architecture, and how these methods can be evaluated.

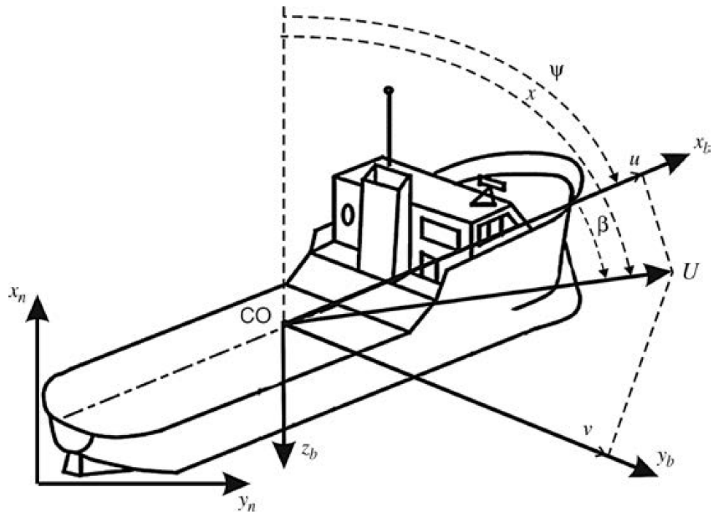


Figure 3.1: The relationship between course (χ) and heading (ψ). The figure is from Thor I. Fossen's book[22].

3.1 Heading Estimation

In many situations the heading of a detected object is crucial information in decision systems. From an object's heading one can extract information about the object's future position or intent. For example, a car heading into an intersection presents a very different situation than a car heading away from one. The heading of an object can be the difference between a critical and a mundane situation. The same is true in the maritime sector, although with some slight differences, since the correlation between heading and course is slightly weaker in maritime environments. Figure 3.1 highlights the difference between course and heading, with the heading defined by the direction of the vessel's bow and the course being defined by the vessel's velocity vector. Under normal circumstances the vessel's heading will be a close approximation of the vessel's course, but in environments with high winds or currents this might not be true.

Most modern sensor systems are limited to estimating an object's course, often through temporal tracking such as maritime radar systems equipped with Automatic Radar Plotting Aid (ARPA). This system tracks a detected vessel's relative position over a time interval, and estimates its course based on its change in measured position. Automatic Identification System (AIS) on the other hand, is able to provide both heading and course information. However, it has some downsides. The first one is that it relies on self-reporting, meaning that the information stems from the detected vessel's own sensor systems. During safety-critical operations it's unfortunate to be reliant on another vessel's sensor system for safety, as its robustness is not known. Secondly, not all vessels are equipped with AIS systems. The Norwegian Coastal Administration has specified what types of vessels are obliged to be equipped with an AIS system. To summarize them quickly, most vessels bigger than a small leisure craft are required to be equipped with one[65]. This means that most small vessels are not equipped with one, and given human nature, some larger vessels probably lack the system as well.

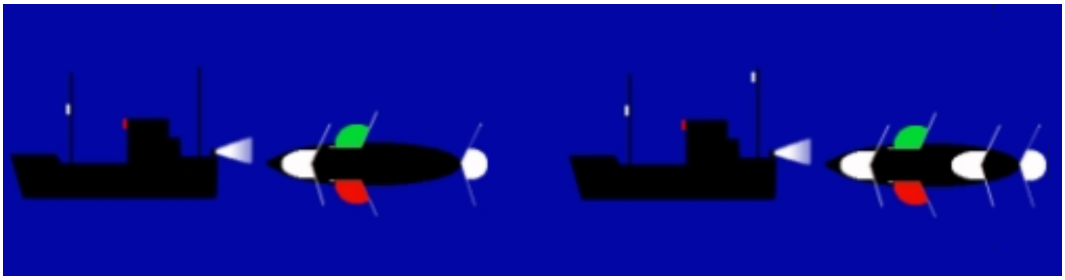


Figure 3.2: Two ships equipped with navigational lights. The leftmost ship is under 50 meters, and the rightmost ship is above 50m and requires a secondary top-light mounted higher than the one mounted in the first mast. The figure is from [23] with some modifications.

3.1.1 Visual Heading Estimation

A natural method for obtaining an object's heading is through visual inspection. As with the automated methods discussed previously in this chapter visual heading estimation has some downsides. For one it is heavily impacted by weather and other visual distractions. During nighttime one is reliant on correct use of navigational lights for accurate visual heading estimation. Often smaller boats[1] do not have adequate navigational lights for visual heading estimation, since they are not required by law[23]. Figure 3.2 shows two ships equipped with proper navigational lights, which allows visual heading estimation during nighttime based on the visible lanterns. The accuracy of visual heading estimation is also limited by the viewpoint. If the observer can't get a good view of the other vessel, either due to occlusion by waves or another disturbance, the accuracy of the heading estimate will suffer. In most cases however, a trained human would be able to provide a reasonable heading estimate.

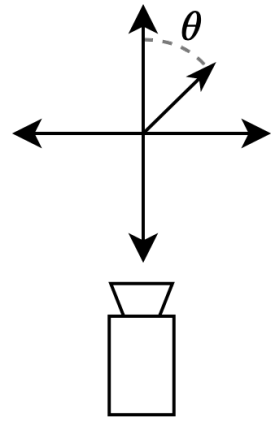


Figure 3.3: An illustration of simple in-plane heading estimation. The task is to estimate the heading of an object (θ) based on a monocular input image.

The machine learning models trained in this thesis are only trained to work in good lighting conditions. That means in daylight and with little or no fog. In a future project it would be very interesting to see how well a network could perform in bad lighting conditions where navigational lights are required for accurate detection, classification, and other predictions. To do this with the current synthetic maritime dataset, information about the vessel's navigational lights would have to be incorporated in their 3D models.

[1] Boats under 7 meters with a maximum speed of 7 knots.

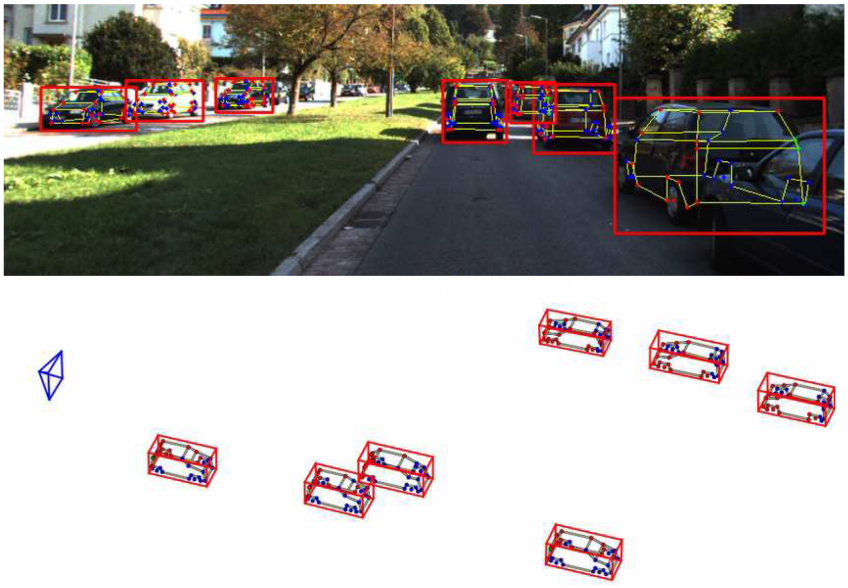


Figure 3.4: Full 3D pose estimation allows the reconstruction of 3D scenes. Here a frame from a monocular camera has been examined by DeepManta, a network for 3D pose estimation debuted in [24].

3.2 Deep Learning Based Methods

There have been several projects aimed at visually estimating the heading of objects with Deep Learning based methods. Most of these have focused on estimating the orientation of automotive vehicles or human bodies. Vehicles are a clear target for such research, due to their implied directionality as well as increased demand from industry as driver assistance systems and self-driving systems become more popular. Before the Deep Learning revolution in the last decade, works such as [66] used more traditional methods like tree-based classifiers to solve the problem. Newer works like [67] opt to instead use convolutional neural networks to perform the task.

There are some notable variations of the heading estimation task. Especially one should separate full 3D pose estimation from simple in-plane heading estimation. The full 3D pose estimation is a more complex task as the number of estimated parameters is higher. How many parameters is estimated depends on how the rotation, position, and camera matrix are encoded, and how the problem is formulated. [68] encodes it in 6 values: the azimuth, elevation, in-plane rotation, distance, and principle point, while [69] include the camera focal length in their estimations, increasing the number of parameters to 7. Figure 3.4 highlights how much information can be extracted from a monocular image by full 3D pose estimation. As is to be expected, there can be large differences between the network architectures employed in the various projects, but most modern systems use a mix of fully connected and convolutional layers in their designs.

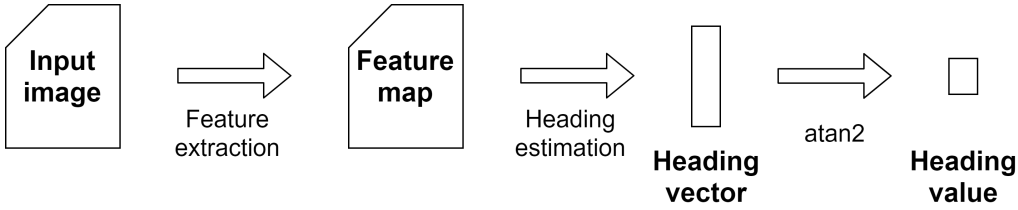


Figure 3.5: The network architecture for estimating an object's heading via a single heading vector.

When it comes to the simplified task of in-plane heading estimation there several possible approaches as well. To clarify, this task is about producing a single angular value representing the detected object's heading in the horizontal plane. Figure 3.3 illustrates this. A paper released in 2017[25], examines three methods of performing this task. This thesis will implement two of them. The third method was not implemented because its loss function only punishes angular heading deviations, and not deviations in magnitude. This was not compatible with the definitions used in this thesis, since some classes are not directional and have their heading vectors defined as the zero vector. The methods tested in this thesis will be discussed in the two next sections.

3.2.1 Heading Estimation as a Regression Problem

The first method described in [25] formulates the heading value regression problem as a heading vector regression problem. This way one avoids the problematic region around $0°/360°$ or $0/2\pi$, where the periodic nature can cause issues. It also means that the magnitude of the heading vector can be seen as an indication of the 'directionality' of an object. If a prediction contains a heading vector with a small magnitude, it may indicate that the object contains little visible 'directionality'. After the heading vector has been predicted, the angular value can be extracted by applying the atan2 function on the predicted vector.

To train a network that predicts a heading vector one can compare it with the unit-vector describing the ground truth direction and optimize with either a L2 loss function or L1 loss function. Figure 3.5 illustrates an architecture that performs heading estimation for a single object through the heading vector regression approach. This approach is simple and intuitive in its design. It is one of three methods evaluated in the experiments later in the thesis.

This thesis will evaluate two variations of heading estimation systems based upon heading vectors. The first one is identical to the one presented in [25], and predicts a single vector representing the detected object's heading. The second one is slightly modified. In this modified version the network predicts one heading vector per defined class in the network. When producing the final prediction vector the network selects the vector corresponding to the predicted class. Such a network is shown in Figure 3.6. The intuition behind this is that the network should be able to use class-specific features in its heading predictions, leading to more precise predictions. On the other hand, it might also reduce the network's ability to utilize strong, general features in its predictions, which would reduce the network's performance. The experiment later in the thesis will examine this.

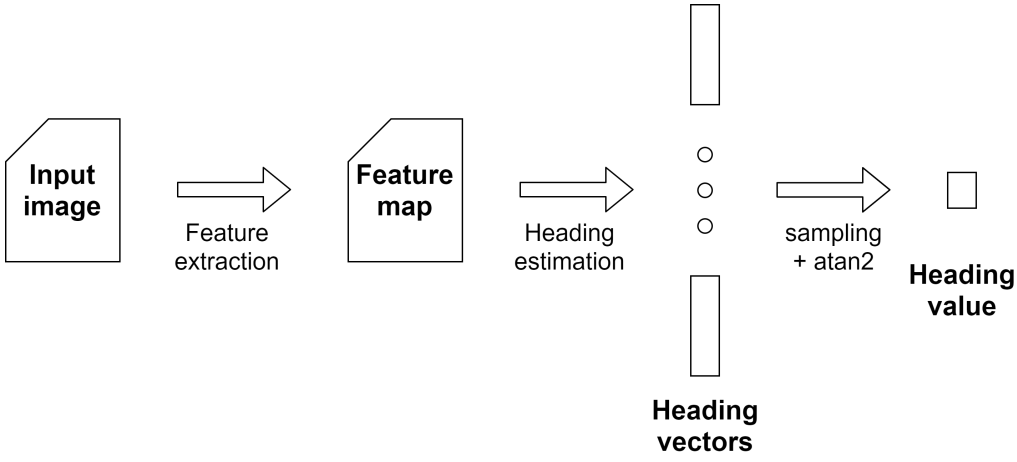


Figure 3.6: The network architecture for estimating an object's heading via several heading vectors.

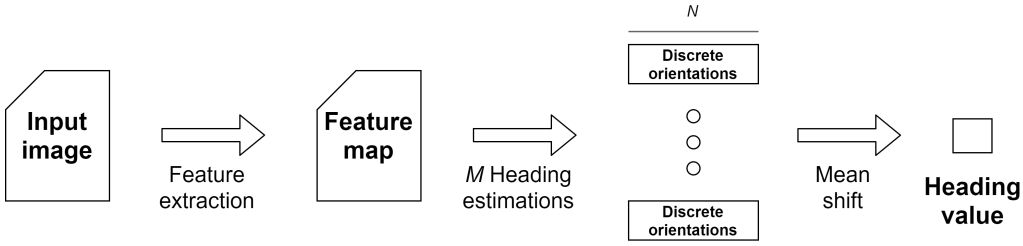


Figure 3.7: The network architecture for estimating an object's heading through combining several classifications. The figure is based on one from [25].

3.2.2 Heading Estimation as a Classification Problem

The third method is a bit more involved. It reformulates the heading regression as several heading classification problems, each with a fixed angular offset. Each classification class then represents a possible discrete heading. Later, the resulting probability distributions are combined into a final regressed heading value through a mean-shift clustering technique. This method is reminiscent of other ensemble based machine learning designs, such as random forest classifiers. The idea is that the collection of classification predictions protect the system as a whole from individual errors. While one of the classifications might work sub-optimally and provide wrong predictions for some inputs, it is unlikely that the error will propagate through to the final result after being combined with all the other predictions. This way the final resulting prediction from an ensemble based design should be more robust and more accurate than a result from a single prediction design.

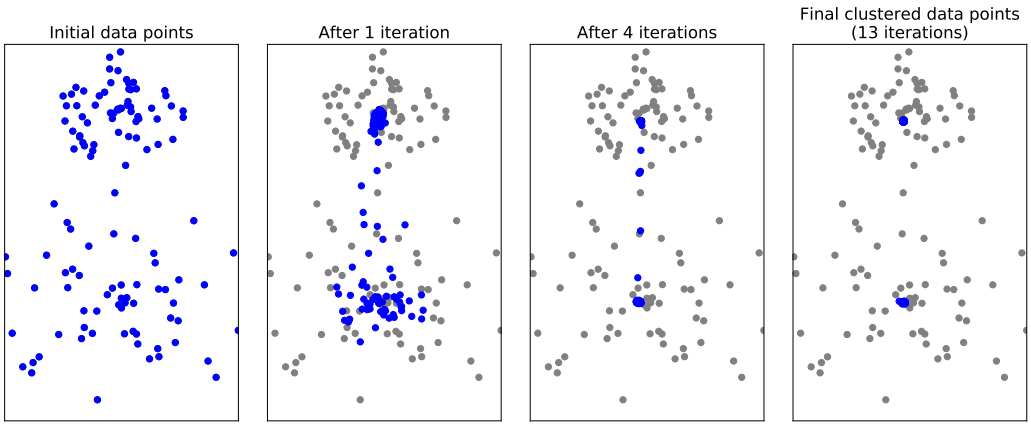


Figure 3.9: Some iterations of a clustering process using mean shift. Observe how the points gradually converge towards the centers of the two clusters. The initial points are shown in grey and the most estimates are shown in blue. The figure was made using a Python script and the Matplotlib module.

As with many other classifiers, the ones in this method are trained with a cross-entropy loss function. For each classification, the discrete heading closest to the continuous ground truth value is set to 1 while the others are set to 0. During the training process all of the classifiers are trained simultaneously, but each with its own angular offset. Figure 3.8 shows how the heading discretizations when using 3 different classifications, each with 4 different discrete classes. Using more classifications results in higher resolution discretizations, but also a more computationally expensive inference process. Increasing the number of discrete heading classifications per classifier also results in a higher resolution discretization, but in addition to increasing the complexity of the computation it also makes each classification task more difficult for the classifiers since the discrete headings within the classifiers are more similar. It is possible to achieve a given discretization level through multiple combinations of classifications (M) and discrete headings per classifier (N), for example (M=1, N=72) or (M=9, N=8). In their experiments, the researchers in [25] found that their networks performed the best when using the ensemble based approach. Figure 3.7 shows a network employing this method.

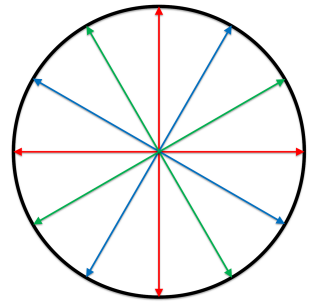


Figure 3.8: The resulting heading discretization when using 3 different classifications, each with 4 different discrete classes. M=3, N=4. The figure is from the paper discussed in this section[25].

So far, how the classification-based approach transforms the set of discrete heading predictions into the final heading prediction value has remained unexplained. As mentioned previously, this is done through a clustering technique called mean-shift. Briefly summarized, the mean shift clustering technique works by iteratively shifting each data point to the average of the data points in its neighbourhood[70]. Figure 3.9 serves as an illustration of this. It shows how the mean shift algorithm detects clusters by gradually shifting the data points towards the center of the clusters. A more

formal explanation follows, based on the one in [70], but shortened to make it more digestible. The curious reader is recommended to visit the original source for a detailed analysis.

Let X be the n-dimensional Euclidian space, R^n, whose components are denoted x_i and whose norm is denoted $\|X\|^2 = \sum_{i=1}^{n} |x_i|^2$ A function $K : X \to R$ is said to be a kernel if there exists a profile, $k : (0, \infty) \to R$, such that

$$K(x) = k(\|x\|^2)$$

and

- k is nonnegative.
- k is nonincreasing: $k(a) \geq k(b)$ if $a < b$.
- k is piecewise continous and $\int_0^\infty k(r)dr < \infty$.

In the mean shift algorithm the kernel function measures the distance between data points and such defines the neighbourhood in which the mean is calculated. Now the sample mean calculation can be defined as

$$m(x) = \frac{\sum_{s \in S} K(s - x)w(s)s}{\sum_{s \in S} K(s - x)w(s)} \tag{3.1}$$

with K being a kernel function, $w : S \to (0, \infty)$ being a weighting function, and S being the entire set of data points. Some variations of mean shift separate the data points being updated into a separate set T, which is independent from S. This means that the calculated mean is based upon the initial data points, which remain constant for the entire clustering operation. By keeping track of the paths the data points have taken during the clustering it is possible to optimize the performance of the clustering operation. The method implemented in this thesis takes advantage of this.

In the case of heading estimation, the weighting function will be the calculated probability of a discrete heading value. Since the space of heading probabilities is circular, the authors of [25] chose to use the von Mises kernel function. This work does as well. The von Mises kernel function is discussed in Section 3.7.3 in [66] and results in the following set of equations for updating the data points

$$\theta^{(s+1)} = \mathbf{m}(\theta^{(s)}, v, \Gamma) \tag{3.2}$$

where $\theta^{(s)}$ is the current estimate of the cluster, $\mathbf{m}$ is the mean shift function, v is a concentration parameter for the von Mises kernel function, and $\Gamma = ((\theta_1, w_1), ..., (\theta_V, w_V))$ is the discrete headings with predicted probabilities. The mean shift function $\mathbf{m}$ is given by

$$\mathbf{m}(\theta^{(s)}, v, \Gamma) = \frac{\sum_{i=1}^{V} g(|\theta - \theta_i|^2)w_i\theta_i}{\sum_{i=1}^{V} g(|\theta - \theta_i|^2)w_i} \tag{3.3}$$

with $g(|\theta - \theta_i|^2)$ equal to

$$g(|\theta - \theta_i|^2) = \frac{v}{2|\theta - \theta_i|} sin(|\theta - \theta_i|)exp(v|\theta - \theta_i|) \tag{3.4}$$

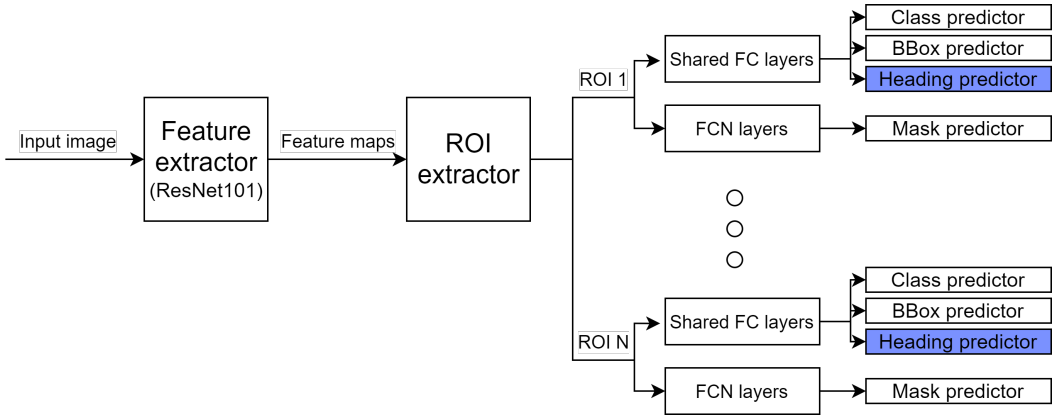


Figure 3.10: The modifications to the Mask RCNN architecture that allows the heading of detected objects to be estimated. The added heading prediction module is colored blue for increased visibility.

Note that because of the circular nature of angular headings, $\theta_i = \theta_i + 2k\pi (k = 1, 2, ...)$, special care must be taken to avoid errors due to not using the smallest possible $|\theta - \theta_i| \leq \pi$. This is to ensure that the algorithm uses the correct, closest distance to other data points in kernel function calculations[66].

3.2.3 Adding Heading Prediction to Mask RCNN

Many possible modifications could be made to the Mask RCNN architecture to facilitate heading estimation. When choosing between the different possibilities one has to regard the inherit trade-off between prediction performance and increased computational complexity. Figure 3.10 shows the modification that was chosen in this work. It involves minimal changes to the existing network structure and simply adds the heading prediction module in parallel with the existing class prediction and bounding box prediction modules. The exact nature of the added heading prediction module depends on the method being tested, it could either be a FC layer producing a 2 dimensional vector representing the heading, one producing a set of $N_{classes}$ heading vectors, one per defined class, or M softmax probability densities with N discrete heading scores. During the development several variations were tested. Both the addition of an extra layer in the heading prediction module and an entirely separate heading prediction system similar to how the mask is predicted was tested. However, none of these variations resulted in clearly better prediction performance during testing and were therefore not used.

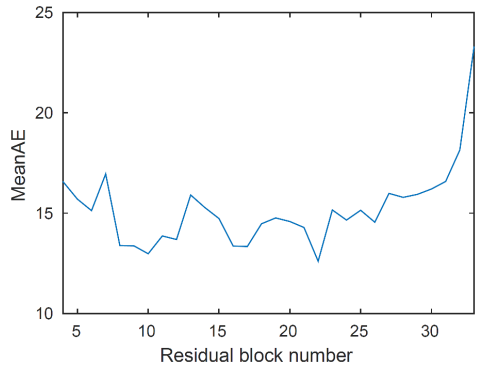


Figure 3.11: The heading performances varies depending on which feature map is used in the calculations. The figure is from [25].

[25], the paper that first presented the heading estimation methods examined in this thesis, raises an interesting point. That intuitively, heading estimation and classification tasks require different types of feature maps. While classification requires rotation invariant features for its prediction since the system should be able to classify an object regardless of its orientation, that is the exact opposite of what a heading prediction system intuitively requires. This could indicate that which feature map is selected as the input to the heading estimation system will heavily impact its performance. The researchers in the aforementioned paper verified this in a simple experiment where they connected the heading prediction module to various levels of the feature extractor. Figure 3.11 shows their results. Their results initially caused concern, since the heading estimation module in this project is connected to the final feature map of the feature extractor in the Mask RCNN network, which should degrade the heading performance according to Figure 3.11. However, as will become clear in the experiment, the heading estimation performance is adequate. This might be because the feature map is trained to facilitate multiple tasks: classification, bounding box regression, and mask prediction, and might therefore already incorporate some rotation dependent features. Keep in mind, this is currently just speculation and research should be done to verify these claims.

Not all of the classes defined in the Mask RCNN architecture need to have a defined direction. It would not be easy to define the direction of classes such as the sky, the ocean, or land. The systems designed in this work also defines wind-turbines as a directionless class. See Section 6.1 for more information about the defined classes. The Mask RCNN architectures implements this functionality in various ways, depending on which method of heading estimation is selected. The first method discussed, which estimates the heading as a heading vector pointing in the same direction as the object, simply defines directionless classes to have the zero vector as ground truth. The second method, which estimates one unit vector per class and samples the one corresponding with the classification, does the same. The third and final method discussed, which reformulates the heading regression problem into a classification problem, simply adds one extra classification value to each classification which represents the directionless case. This additional value is removed and the rest of the classification values are normalized before the mean shift algorithm determines the final predicted heading value.

3.3 Evaluating Heading Estimation

As with the other aspects of Deep Learning it is important to have a precise framework for measuring the performance of a model. In the case of heading estimation, the measurements should provide insights into how often the model predicts accurate headings with regards to the ground truth. In the case of simple in-plane heading estimation there are typically three measurements made to access the performance of a model. The first is the Mean Absolute Error (meanAE), which measures the mean absolute angular prediction error for a given set of predictions. Naturally, an accurate model exhibits a small meanAE. The second measurement is medianAE. It is similar to the previously mentioned measurement, but instead of calculating the mean value of the absolute angular errors it calculates the median value. As other median value measurements it is more resistant to high value outliers and might therefore capture a more 'realistic' picture of the model's performance. The third measurement is simply to measure what percentage of the predictions fall within a defined angular

distance from the ground truth angle. A well performing heading estimation system should provide predictions that fall within a close angular distance of the ground truth distance, so the accuracy percentages should be close to 100%. This kind of measurement will be noted int the text as Accuracy+Threshold. For example: Accuracy22.5, which describes what percentage of the predicted headings fall within 22.5° of the ground truth heading.

Chapter 4

Explainable Artificial Intelligence

This chapter explores the field of XAI, gives a quick introduction to its history, and highlights some of its main challenges. It describes the advantages of creating XAI systems, both for the end-user and the system creator. It also explores what constitutes a good explanation, how one should be delivered, as well as what information one should contain. The topic of discriminative attributions are also covered. Finally, various methods for constructing interpretable machine learning algorithms and how they can be adapted to use for instance segmentation are discussed.

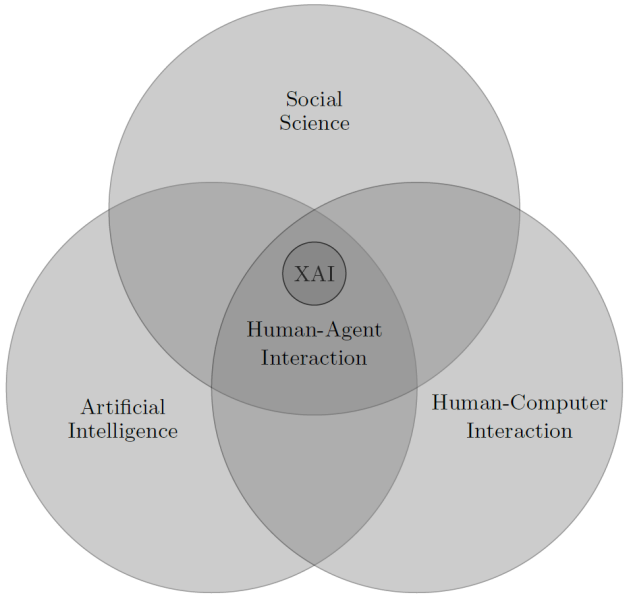


Figure 4.1: [26] describes XAI as influenced by several research domains. This figure is from that paper and shows a venn diagram illustrating this concept.

The term XAI was first coined in a paper released in 2004 by Michael van Lent[71]. It has since become the de facto term used when describing AI systems that attempt to explain their own behaviour. XAI is a new field and unfortunately not many rigid definitions exist. The problem however, has been around for some time. In the 1970's, AI systems primarily consisted of expert systems. These were decision making programs that utilized hierarchical if-then structures to perform inference. The if-then structure was usually programmed by a domain expert using the programmer's extensive domain knowledge. Explanation of these systems was subject to research, but was of a different nature because of the expert systems hierarchical structure.

As the interest in expert systems died down over the following decades, so did the interest in XAI. Then, in the 2010's, the Deep Learning revolution led to a large increase in complex machine learning models. This increase has caused a revival of interest in XAI. Now that these complex machine learning methods are to be applied in safety-critical operations XAI is poised to play a key role.

There are two commonly used terms in XAI research: explainability and interpretability. They are used interchangeably, an unfortunate result stemming from the lack of clear definitions. One definition of interpretability is "the ability to explain or to present in understandable terms to a human"[72]. This defines explainability as a part of interpretability. More specifically as the method for transferring information about the causal events in the AI system to the receiving human. That is how this thesis will relate to these two terms.

XAI is a problem spanning several fields of research. [26] attributes three contributing fields

.

Figure 4.2: An explanation of a husky-wolf image prediction. The input image is shown on the left and the segments provided as an explanation is shown on the right. The image was classified as a wolf. The segments highlight the areas of the image which influenced the classification the most. The figure is from [27]

to XAI: AI, Human-Computer Interaction, and Social Science. A Venn diagram describing this is shown in Figure 4.1. This multi domain split further contributes to the difficulty in assigning proper definitions. Researchers from each of these fields have separate points of view caused by their different academic backgrounds. This leads to various definitions and methods being used in research.

4.1 Why Create Explainable AI Systems?

If an AI system could explain its decision process in an understandable manner it would provide several advantages. Some of these advantages will be highlighted in this section.

For safety-critical applications the creator of the AI algorithm must be able to prove the reliability of the system. Even though the current solution to many safety-critical problems: humans, are not always perfect, they are able to explain their decisions which allows the system to improve after an error has occured. A Deep Learning based machine learning system operates as a black box without available explanations. This makes pin-pointing the cause of an error difficult, if not impossible. This can make it difficult to improve such a system after an error. In order for legislators to accept autonomous vehicles such as autonomous ships, the creators must be able to justify their safety. XAI might be the technology that could enable the creators to do this.

In [27] the authors show a good example of why XAI is valuable. In their experiment they trained an image classifier to separate images of huskies from images of wolves. Their classifier performed well - it successfully separated the images by subject. The researchers then used an XAI method called LIME to generate explanations for the classifications. One of these explanations is shown in Figure 4.2. A quick glance at the image segments provided in the explanation allows even a non-expert to deduce that the classifier uses the snow on the ground to classify the image, not the

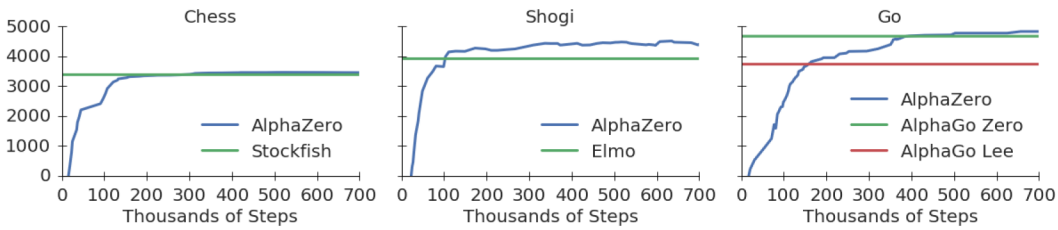


Figure 4.3: The ELO of the AlphaZero algorithm during training. Note that it exceeded the best previous algorithms. The figure is from [28].

features of the animal. This indicates that the model will not work well in the real world, in which images of wolves might not necessarily contain snow. It is clear that an explanation like this makes it easy to validate the model or spot erroneous behaviour.

Building on the husky-wolf classifier example above, one can easily imagine how these XAI methods can be used to improve a model. By examining the explanations for the dog/wolf classifier, the creator would probably spot the model's bias quickly. This bias could then be removed by expanding the dataset to include images of wolves without snow present in the background. The performance of the network during training would probably go down, since the network can no longer base its decisions purely on the presence of snow. But the real world performance would likely improve dramatically.

The end-user's trust in the AI system is also important. In many professions classifiers like the ones discussed above could be a great tool to have. As these systems become more advanced the users will likely rely heavily on them. This begs the question: who is to blame if the prediction is wrong? In some professions a wrong prediction could have large consequences and the user would likely end up with a portion of the blame. If the classifier could provide explanations along with its predictions, the end-user could more effectively evaluate their quality. This would not only improve the end-user's trust in the system, but could also avoid potentially damaging errors.

[73] raises another interesting point, that XAI will enable us to discover new information from machine learning algorithms. They use the AlphaZero algorithm, published by DeepMind in 2017, as an example. The AlphaZero algorithm teaches itself to play board games entirely without human intervention. By doing this it has superseded all previously known algorithms in three different board games, as seen in Figure 4.3. Maybe new thought-processes and/or play-styles could be discovered if AlphaZero could explain its record breaking tactics? That would undoubtedly be fascinating, but maybe not immediately useful, as chess tactics are of limited use outside the game. But what if this algorithm was instead trained to treat dangerous illnesses? Then a good explanation could potentially save millions of lives.

4.2 What Is a Good Explanation?

A natural follow up question is what constitutes a good explanation. Unlike many other questions within computer science, this question does not have a clear answer. It is a very subjective question and all this section can do, is offer some general guidelines that can help when creating explanations in the context of XAI systems.

[26], a survey paper released in 2017, suggests that contrastive explanations are most effective. By this they mean that for a human to best understand an explanation, it must provide examples of where the causal event did not occur. For instance: if an explanation wants to convey that event A causes event B, it must show examples where event A does not occur and as a result event B does not occur either.

A good explanation should be adapted to the receiver's knowledge level. For instance: if the receiver is a programmer with intimate knowledge of the AI system, the explanation can be very technical and still be helpful. But if the receiver is a lay-person, it must be formulated in an easy to understand way. Naturally this is not always possible, if there is a technical problem with the AI system, the XAI method might be forced to give a complex technical explanation. In that case the end-user will have to call for technical assistance.

[74] hypothesises that humans use internal mental simulation models when making predictions. To answer "what-if" questions, we change the initial conditions of the relevant mental model, and observe how the mental simulation outcome differs. A good explanation should be adapted to the receivers internal model of the system. It should show abnormal causes, that means causes that are not already included in the receivers mental model. The receiver can then modify his/hers internal model and learn something new. This is equivalent to generating explanations which take the confirmation bias into account. By taking the confirmation bias into account, one combats the human tendency to find solutions that fit their mental model of the world, while ignoring solutions that do not. By explicitly showing examples that counteract this mental model, the receiver would be forced to change his/hers internal model. Naturally, this begs the question of how the XAI method can determine the receiver's mental model.

In the real world an event often has countless causes. It is the XAI system's role to filter through them all and only deliver the most important ones to the receiver, according to the guidelines defined above. In computer vision the problem is somewhat simplified, since the possible causes are often limited to distinct features of the input image. Still, generating intuitive explanations for classifiers is no easy task. Some methods for doing this will be presented in Section 4.5.

4.3 Discriminative Attributions

A concept that might play a large role in future XAI research is discriminative attributions. Briefly summarized, a discriminating attribute is an attribute that separate one concept from another concept. As an example, the attribute *flying* separates the concept *plane* from the other concept *fence*. The detection of discriminative attributions is closely related to semantic similarity, which measure

the 'distance' between terms based upon their semantic likeness or their meanings. To illustrate: a *boat* is semantically similar to *canoe*, even though the words are very different.

In the 2018 SemEval competition featured discriminative attributions in one of its tasks. When the competition was finished they also released a paper[29] discussing the task, the datasets, and the submissions. In the paper they argue that research into discriminative attributions is necessary because no system that calculates semantic similarity has really understood the problem if it cannot also report why two terms are different. They use the coffee drinks *americano* and *espresso* as examples and express a desire for systems that are able to understand that an *americano* is bigger than an *espresso* and/or that the *espresso* drink contains milk foam. In their challenge they define the task of evaluating discriminative attributions as a binary classification problem with three inputs (*term1*, *term2*, *attribute*), and request systems that can predict whether the attribute is an discriminative attribute in relation to the two terms.

As with many other fields within XAI, discriminative attribution detection also suffers from a lack of clarity and definitions. The main problem is that deciding whether an attribute is discriminative is a subjective task. As a result of this, a number of the samples in the SemEval-dataset was labelled differently by different people. The paper highlights *garlic* and *wings* as an example. *Garlic* is related to *wings*, not because garlic has wings, but because garlic chicken wings are a popular dish[29]. Examples like this show why making systems that evaluate discriminative attributions is difficult. Figure 4.4 shows some example data samples from the SemEval paper[29].

$word_1$	$word_2$	$attribute$
airplane	helicopter	wings
bagpipe	accordion	pipes
dolphin	seal	fins
gorilla	crocodile	bananas
oak	pine	leaves
octopus	lobster	tentacles
pajamas	necklace	silk
skirt	jacket	pleats
subway	train	dirty

$word_1$	$word_2$	$attribute$
tractor	scooter	wheels
crow	owl	black
squirrel	leopard	fur
pillow	jacket	white
dresser	cupboard	large
spider	elephant	legs
gloves	pants	wool
gorilla	panther	long
scarf	slippers	colours
lion	zebra	large

Figure 4.4: Example discriminative attributions from the SemEval paper[29]. The first box contains discriminative attributions and the bottom one contains attributions that are not discriminative.

[75], published in 2019, further analyses discriminative attributes and highlights a handful of different types. The following explanations are from that paper, with some small changes.

- **Essential vs. Incidental**: *p* is an essential attribute of object *o* if it is necessary that *o* has *p*, otherwise it is an incidental attribute. As an example, (*car*, *bus*, *Wheel*) is an essential attribute while (*car*, *bus*, *Gasoline motor*) is an incidental attribute.

- **Sensory vs. Logical**: *p* is a sensory attribute of an object *o*, if *p* can be identified as a property of *o* exclusively through sensory attributions, if it cannot it is a logical attribute. As an example, (*car*, *bus*, *red*) is a sensory attribute while (*car*, *bus*, *vehicle*) is a logical attribute.

- **Relative vs. Absolute**: p is a relative attribute of an object o_a in relation to o_b just in case there exists an object o_c with the attribute p where p is not an attribute of o_a in relation to o_c. On the other hand, p is an absolute attribute of an object o_a in case there does not exist an object o_b with the attribute p where p is not an attribute of o_a when compared to o_b. As an example, (*giraffe*, *ostrich*, *tall*) is a relative attribution, while (*bat*, *butterfly*, *fur*) is an absolute attribution.

It is obvious that a system able to express discriminative attributions in relation to its predictions would be able to offer a high level of interpretability. However, at the moment, the state-of-the-art methods are quite limited in their abilities. The systems will likely have to be expanded beyond text and modified to work with other inputs such as images, before they would be helpful in computer vision applications.

4.4 Interpretable Machine Learning

Before venturing into the concrete methods for interpretable instance segmentation, the main methods for designing interpretable Machine Learning (ML) systems will be covered.

There are two main methods for creating interpretable machine learning systems. One method is to simply use models that are intrinsically interpretable. This can be achieved by using a hierarchical architecture or by using a sufficiently simple model. As an example, one could design a cat-detector model that detects the features of the cats separately, before combining them into a final cat detection. This model could explain itself by simply highlighting the relevant parts of the cat that led to the classification. But the problem remains - how would the system explain why it detected an ear at a specific location? The modularity of the system would have to be quite fine grained to offer satisfactory interpretability. An example of the alternative solution would be using a cat detector algorithm simple enough that a human could directly understand its decision making process. The operator could then just interpret the internals of the model and deduce the explanation.

There is one downside to intrinsically interpretable models. Unfortunately, the simpler models tend to produce less accurate predictions. There is an intuition to this: a more complex model can detect more complex patterns in the input data, but is inherently less interpretable. To make a model more interpretable it would have to be less complex, but this might prevent it from detecting the complex patterns it would otherwise have been able to base its predictions on. For some applications this is not a problem, the performance loss is acceptable. But for more advanced machine learning tasks, such as safety-critical instance segmentation in maritime environments, the performance loss is unacceptable and other methods must be used. This trade-off is called the interpretability-performance trade-off. Recently there have been attempts to merge high-performing neural net based designs with simple and interpretable decision tree based designs. As a quick reminder, a decision tree machine learning system performs inference by performing a series of branching tests on the input. These branching tests resemble a tree structure and when the input reaches a leaf node, a prediction is made based on which leaf node the input reached. This structure is inherently interpretable because it allows direct insight into the thought process of the model, but as previously discussed, its simple decision making process prevents decision tree based methods from reaching the same performance

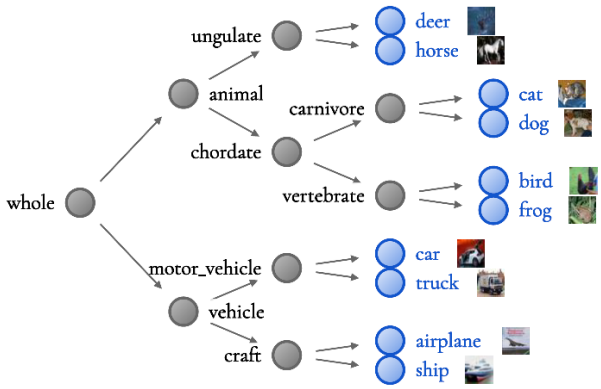


Figure 4.5: The induced hierarchy with linguistic labels from a WideResNet model trained on the CIFAR10 dataset. The figure is from [30], with some modifications.

levels as modern neural net based networks such as ResNet. One recent example of a successful merging of neural networks and decision trees is shown in [30], in which the authors define a hierarchy from a neural network already trained on a dataset before fine tuning the network weights with a loss function customized to the induced hierarchy. During inference, the neural network backbone constructs a feature vector which is iteratively compared to feature vectors representing each node in the decision tree, until a leaf node is reached and the corresponding class is chosen as the prediction of the network. To further increase interpretability the authors also use a WordNet based hierarchy to label each decision linguistically. Their resulting system achieved performance comparable to modern neural net based methods, but with much higher interpretability. Figure 4.5 shows a hierarchy induced by this method.

The second method for achieving interpretability is through ad-hoc methods. These methods do not modify the existing ML system. Instead they systematically modify the input data while observing the outputs to learn about the behaviour of the model. These methods have several advantages over the intrinsic methods. For one they are not subject to the interpretability-performance trade-off since they do not modify the model being examined. They are also often model-agnostic. This is a term used when an XAI method can be used on all ML models. This flexibility is of course desirable. A downside is that these methods are inherently less ”elegant” than the intrinsic methods, since they add extra systems around the original model. They can also be computationally expensive because of their need to observe the input-output relationship of the models they explain, this could prevent such systems from being used in real time applications.

There is a distinction between model explanations and prediction explanations, also called global explanations and local explanations. Model explanations convey how the different parts of the model influence decisions. In other words: how the model works. These methods are usually model specific since they use the specific features of the model in their explanations. Prediction explanations aim to inform why a model reached a specific decision. For instance: a sailboat detector might highlight the sail in the image as an explanation.

An interesting method of achieving interpretable machine learning is via surrogate models. In short, these methods train a separate machine learning model to interpret the first model. This might seem counterintuitive at first: why use more machine learning when we already struggle to interpret one system? Well, adding a secondary ML model can actually improve the interpretability if the secondary model is designed correctly. The XAI method LIME does this by training a ML model to explain one of the primary models predictions. It then uses the parameters of the trained interpretable secondary model to infer which parts of the input strengthens the main model's prediction and which inputs weaken it[27]. This makes LIME a local surrogate method.

In contrast to local surrogate methods, a global surrogate model method attempts to train an interpretable surrogate model to predict the entire decision process of the primary model. This is generally a more challenging task, since the surrogate model often cannot represent the whole model accurately over the whole input-output space. If it could, one would just use the surrogate model instead of the more complex primary model. Both local and global surrogate model methods are often model-agnostic.

Some XAI methods tries to highlight which parts of the input is the most important for a prediction result. These are inherently local explanation methods since they operate on an individual prediction. This can be done by modifying the different features of the input, and recording the change in output. Intuitively, altering the most important features will lead to the greatest change in the output. A method called Integrated Gradients employs a similar method for highlighting the most important parts of an image[32]. Integrated Gradients is one of the methods that will be further discussed in Section 4.5.

4.5 Interpretable Instance Segmentation

The Mask R-CNN instance segmentation method described in Section 2.3.5 is a multiple stage process. This complicates the implementation of XAI methods, so to facilitate easier explanations, the following sections will use a simple image classifier network in the discussions. The general principles of each method translates between the two use-cases. Image classification is a simplification of the object detection algorithm illustrated in Figure 2.1a. It does not predict bounding boxes around the objects and is limited to one detection per image.

Humans are visual creatures. Therefore, a natural way for a computer vision algorithm to explain its predictions is to highlight the parts of the input image that most influenced the result. There are other ways to gain insight into the model; an alternative method could be to find an image that maximize a class prediction score and use that image to explain which patterns the model is looking for. This could be one way to generate explanations for objects that are not defined by a handful of visual features, such as the sky, or the ocean. One downside to these purely visual explanations is that they do not incorporate information about the internal decision process of the instance segmentation model, since they simply highlight features of the input image. This lack of information means that the receiver of the explanation must estimate the model's internal decision process on his/her own. This estimation might be inaccurate or even completely wrong. This might degrade the model's true

interpretability, or just confuse the receiver.

Feature attribution is a research field that explores how computer vision algorithms see the world. It does this by highlighting the components of the input image that most affect the output prediction. For example: when predicting the class "sailboat" for an input image, the model should present the sail as an important feature. If it does not, it could indicate that the model does not generalize well, since it does not understand the defining feature of the sailboat class. Saliency maps are closely related to feature attributions; they highlight the areas that most influence the prediction on a pixel by pixel level. Two of the methods that will be evaluated in this thesis produce such saliency maps. Feature attribution is closely related to discriminative attributes, which is discussed in Section 4.3. To expand on the earlier example, the sail of the sailboat would be a discriminative attribute between the sailboat class and the other defined classes in the model, since likely only the sailboat class would have a sail. An effective image classifier model would be expected to learn this and use it in its classifications. Because of this, feature attribution could be viewed as a kind of visual discriminative attribution detector. With the exception that it highlights which attributes the model has learned to be discriminative, which might differ from the attributes that humans determine to be discriminative.

4.5.1 Jacobian Matrices

The simplest type of saliency map is the Jacobian matrix, it describes the derivative of a function's output with respect to its inputs. An image classifier can be seen as a function

$$f : \mathbb{R}^n \to \mathbb{R}^m$$

that maps inputs with n number of dimensions to outputs with m number of dimensions. The derivative of this function with respect to the input variables is called the Jacobian.

$$\mathbf{J} = \begin{bmatrix} \frac{\partial \mathbf{f}}{\partial x_1} \cdots \frac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} \cdots \frac{\partial f_1}{\partial x_n} \\ \vdots \ddots \vdots \\ \frac{\partial f_m}{\partial x_1} \cdots \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

If a classifier network takes as input an RGB image with 10^6 pixels and classifies the images with regards to two possible classes the classification function will be equivalent to

$$f : \mathbb{R}^{3 \times 10^6} \to \mathbb{R}^2$$

if the output values of the classifier represent the probability of the input representing a defined class. The Jacobian matrix can be estimated numerically by the approximation

$$\frac{\partial f}{\partial x_i} = \lim_{h \to 0} \frac{f(x_{i0} + h) - f(x_{i0} - h)}{2h}$$

where x_{i0} is the x value where the derivative is to be calculated and h is a sufficiently small constant. Many Deep Learning software programs also have gradient calculations build in. In the case of an image, x_{i0} would be the value of a given RGB channel of a pixel in the input image. It

Figure 4.6: [31] used the Jacobian matrix of an image classifier to generate the feature attribution shown above. This figure is from that paper.

is now clear that the Jacobian matrix encodes how small perturbations in the input would impact the output. Intuitively, the pixels with the highest corresponding value in the Jacobian matrix are the most important pixels in the image, since changing them has the largest effect on the output.

An XAI method based on interpreting the Jacobian matrix would have some weaknesses. One being that it is a linear estimation of the function at a given position in pixel-space, and it is only valid for small perturbations since it does not predict non-linear effects. This is unfortunate, since most image classifiers are highly non-linear mappings from the input pixels to the output predictions. Also, since the Jacobian matrix is generated by perturbing one pixel at a time, it does not take into account how the output might change if several pixels are perturbed at the same time. In real world images the pixel values are expected to be highly correlated. In an image, a ship's feature might cover several pixels and require large scale changes in order to disappear into the background texture of the image. The Jacobian matrix will not necessarily describe the effect of these changes accurately. This limits the Jacobian matrix's ability to highlight the important object components present in an image. Still, as shown in Figure 4.6, the attributions it generates can still offer some insight into a model.

4.5.2 Integrated Gradients

In many ways the Integrated Gradients method is a natural extension of the Jacobian matrix method. It works by defining a baseline input $\mathbf{x}'$ in addition to the input image $\mathbf{x}$. The method then gradually steps the input from the baseline values ($\mathbf{x}'$) towards the actual inputs ($\mathbf{x}$). The baseline is meant to represent no input, but since there is no pixel value representing that situation, black or the average image color is often used. By formulating the problem like this, Integrated Gradients incorporates a form of counterfactual intuition. This is similar to how humans often assign causes by examining a

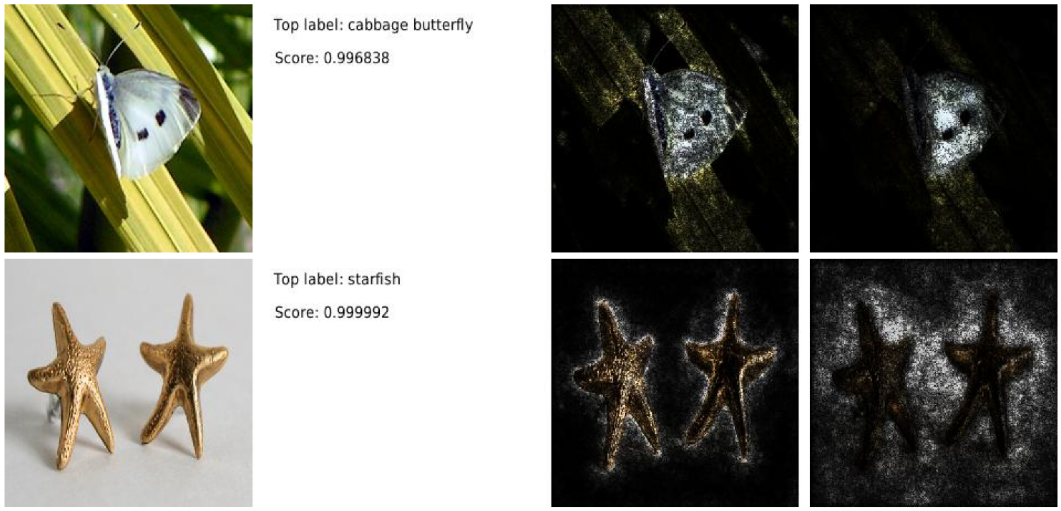


Figure 4.7: The resulting attributions after applying Integrated Gradients to a set of predictions. The Integrated Gradients result is shown to the left, while the Jacobian method is shown to the right for comparison. The figure is from [32].

similar situation with the relevant feature absent[32].

Integrated Gradients considers a straightline path from $\mathbf{x}'$ to $\mathbf{x}$ and computes the gradient at all the locations. Other paths than the straightline path could be conceived, the attribution methods that perform a path integral from some baseline input to the final input are collectively known as path methods[32]. The pixel-wise attribution to the output is then defined as the sum of these gradients. It is clear that Integrated Gradients is really just an extension of the Jacobian examination method. Only instead of calculating the gradient once, it does it repeatedly while changing the inputs. Mathematically the integral along a dimension (a pixel's RGB value) is defined as

$$IG_i(\mathbf{x}) := (\mathbf{x}_i - \mathbf{x}'_i) \times \int_{\alpha=0}^{1} \frac{\delta F(\mathbf{x}' + \alpha \times (\mathbf{x} - \mathbf{x}'))}{\delta \mathbf{x}_i} d\alpha \tag{4.1}$$

where α is an interpolation variable and $\frac{\delta F}{\delta \mathbf{x}_i}$ is the derivative of the output with respect to a dimension (a pixel's RGB value)[32]. This integral can be approximated by a series of sums

$$IG_i(\mathbf{x}) \approx (\mathbf{x}_i - \mathbf{x}'_i) \times \sum_{k=1}^{m} \frac{\delta F(\mathbf{x}' + \frac{k}{m} \times (\mathbf{x} - \mathbf{x}'))}{\delta \mathbf{x}_i} \times \frac{1}{m} \tag{4.2}$$

in which m represents the number of steps used in the approximation[32]. Figure 4.7 shows two attributions generated with this method, as well as two generated by the Jacobian matrix method for comparison.

This method improves on the Jacobian matrix attribution method by examining the path from the baseline image to the input image. By doing this it encodes not just the local effect of perturbations,

but how perturbations affect the output for a various of inputs. This should provide a more robust feature attribution. The more robust feature attribution does however, come with a computational cost. Integrated Gradients calculates the gradient multiple times when estimating the path integral. In practice it was found that between 20 and 300 steps provided a good result[32]. This means that the Integrated Gradients method is at least 20-300 times slower than examining the Jacobian method described in the previous section.

Both the Jacobian method and Integrated Gradients work on a pixel by pixel basis. This means that they do not encode how changing combinations of pixels affects the output. To achieve this, the algorithm must work on super-pixels instead of individual pixels and analyse how changing batches of pixels affects the output. On the other hand, the method described in the Section 4.5.4 does just this.

4.5.3 Integrated Gradients and Heading Regression

This section details how the Integrated Gradients method can be modified to create feature attributions explaining a detected object's estimated heading. In the previous section, the discussion mainly dealt with how Integrated Gradients is used to explain a classification result by repeatedly calculating the gradient of the classification score with respect to the input pixels as the input image is shifted from a baseline image to the actual image. Exactly how to modify Integrated Gradients to work with heading estimations instead of classifications depends on how the heading estimation is performed. Since the feature attribution experiment in this thesis is performed on a network which estimates the heading of a vessel through a vector, that will be the focus of this section. The details surrounding this estimation can be found in Section 3.2.1, and details about the experiment can be found in Section 6.3.

This thesis implements a pragmatic approach to adapting the Integrated Gradients method. By analysing the problem it quickly becomes clear that the only difference is that the networks output two values, the two components of the unit vector representing the heading, instead of the single classification score. This means that we would have two sums, since Equation 4.2 is applied twice, on both values of the unit vector. In the end we would have two values per RGB channel per pixel in the image, totalling 6 values per pixel. To keep the modified method as similar as possible to the original Integrated Gradients implementation and attempt to extract information about the directionality of the derivatives with respect to the ground truth heading, another method was used. It instead merges the two derivatives through a special merging function before adding them all together in the sum defined in Equation 4.2. This leads to the following equation describing the system

$$IG_i(\mathbf{x}) \approx (\mathbf{x}_i - \mathbf{x}'_i) \times \sum_{k=1}^{m} G\left(\frac{\delta F_x(\mathbf{x}' + \frac{k}{m} \times (\mathbf{x} - \mathbf{x}'))}{\delta \mathbf{x}_i}, \frac{\delta F_y(\mathbf{x}' + \frac{k}{m} \times (\mathbf{x} - \mathbf{x}'))}{\delta \mathbf{x}_i} \right) \times \frac{1}{m} \quad (4.3)$$

where $\frac{\delta F_i}{\delta \mathbf{x}_i}$ is the derivative of a value in the output unit vector with respect to a dimension (a pixel's RGB value), m represents the number of steps used in the approximation, and G is the merg-

ing function. Figure 4.8 illustrates the two derivatives of the predicted heading vector.

The natural next question is how to define the merging function. This thesis proposes three different functions. The first and simplest method is to simply to merge the two derivatives using the L2-norm of the vector created by combining the two derivative values per pixel RGB value. This can be written as

$$G_{L2}(dF_x, dF_y) = \sqrt{dF_x^2 + dF_y^2} \tag{4.4}$$

with dF_x and dF_y representing the values of the derivatives. This merging function was initially preferred due to its simplicity, but it was unclear whether it would provide clear feature attributions since it does not take into account the actually predicted heading of the object. Intuitively, one should take into account the actually predicted heading vector when performing this merging operation, since not all derivatives impact its angle as defined by the atan2 function. This intuition inspired the next two merging functions.

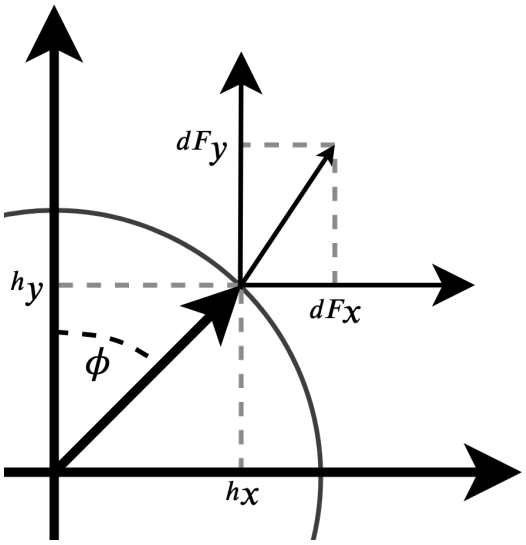


Figure 4.8: The heading unit vector, represented by the thick arrow, results in two derivatives, dF_x and dF_y. ϕ represents the predicted heading while h_x and h_y represents the components of the predicted heading vector.

The next two merging functions include the actually predicted heading and use it to separate the gradient vectors that actually induce a change in the predicted heading from the ones that do not. They both work by decomposing the gradient vector with respect to the predicted heading vector. The first merging function decomposes the gradient into the parallel component via

$$G_{parallel}(dF_x, dF_y) = \left\| \frac{(h_x, h_y) \cdot (dF_x, dF_y)}{h_x^2 + h_y^2} (h_x, h_y) \right\| \tag{4.5}$$

where dF_x and dF_y represent the values of the gradient while h_x and h_y represents the values of the predicted two dimensional heading vector. This way only gradients that contribute to the predicted heading are included after the merging of the two gradient values. In the second method only the orthogonal components are included through

$$G_{orthogonal}(dF_x, dF_y) = \left\| (dF_x, dF_y) - \frac{(h_x, h_y) \cdot (dF_x, dF_y)}{h_x^2 + h_y^2} (h_x, h_y) \right\| \tag{4.6}$$

which should filter the gradients so primarily the gradients that induce a change in predicted heading direction are retained in the feature attribution.

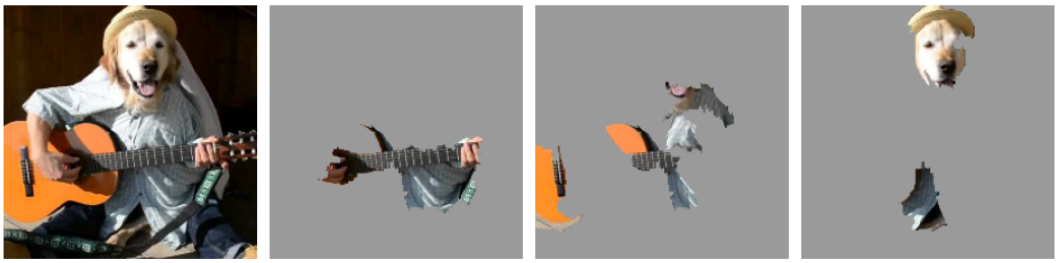


Figure 4.9: Shown above are three explanations generated for an image classifier by LIME. To the left is the segments that cause the model to classify the image as *Electric guitar*, in the middle are segments that induce *Acoustic guitar*, and to the right are segments that leads the model to classify the image as *Labrador*. The figure is from [27].

4.5.4 Local Interpretable Model-Agnostic Explanations (LIME)

LIME uses a different approach than the two saliency map methods described so far. Its goal is to generate an interpretable surrogate model that is locally faithful to the classifier[27]. Locally faithful means that the surrogate model correctly represents the original model for small changes to the input image. Another difference is that it does not perturb the input image on a pixel by pixel basis, but rather on a segment basis. This means that LIME requires an image segmentation algorithm that divides the input image into segments that LIME can then use for its training process. As seen in Figure 4.9, LIME's explanations are similar to the saliency maps, but instead of highlighting the features on a pixel by pixel basis it highlights different segments of the images. Since it trains a surrogate model on the perturbed input image it also encodes higher value information than the simpler saliency maps methods, as seen by clearer results.

The LIME algorithm works by minimizing a locality-aware loss $\mathcal{L}(f, g, \pi)$ with f being the classifier, g being the interpretable surrogate model, and π being a similarity measure between the input and the sampled perturbed inputs. More specifically LIME defines $x \in \mathbb{R}^d$ as the input and $x' \in \{0,1\}^{d'}$ as the interpretable representation of that input. For images x' represents the presence or absence of the image segments. Imagine that the image is segmented into N segments, then $x' \in \{0,1\}^N$, represents whether the image segment is present or replaced by some baseline color.

LIME samples z' in the vicinity of the input x' and classifies these with the original model to create a dataset for the surrogate model. It then determines which subset of the x' features best describe the original model's output and trains the surrogate model to predict the original model based on these features. Note that the selected perturbed samples z' are weighted according to the similarity measure π, since their prediction might differ due to non-linear effects in the original model. Since the LIME surrogate model is a simple model, it won't be able to represent these non-linearities. By reducing the effect of these perturbed samples, the model becomes more robust to this sampling noise. This is shown in Figure 4.10, where LIME trains a linear surrogate model on a two dimensional classifier. The pink and blue background represents the original classifications, while the crosses and the circles represent the perturbed samples z'. The size is proportional with the simularity measurement and the dotted line represents the resulting linear surrogate model. Note

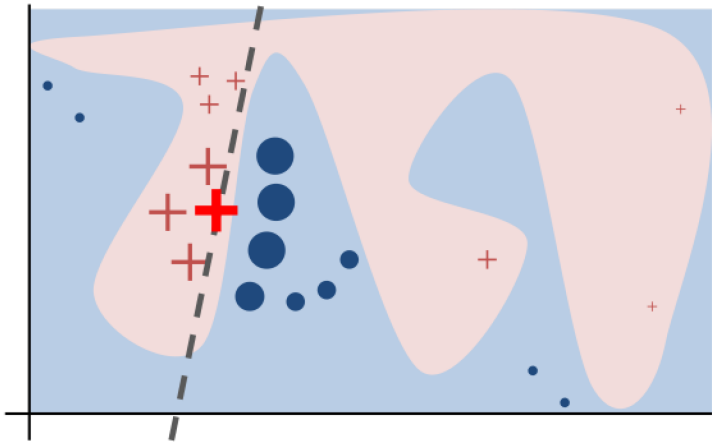


Figure 4.10: This figure shows the sampling of perturbed z's for a simplified two dimensional model. The figure is from [27].

that it is locally representative for the original model, but fails to represent it for the entire state space.

The LIME algorithm does not explicitly specify what kind of surrogate model is used. In LIME, a linear classifier is used. This defines a weight vector $\mathbf{w} \in \mathbb{R}^m$ where m is the number of features selected to train the surrogate model. The classification is then defined as

$$g(\mathbf{z}') = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{z}' > T, \\ 0 & \text{otherwise} \end{cases}$$

where T is some threshold value. When training, the values of $\mathbf{w}$ and T are gradually shifted towards values optimizing some defined loss function. In the LIME paper they use L2 loss with the similarity measurement as weight. Because the values of the $\mathbf{w}$ vector directly measure how much a feature impacts the prediction, LIME uses them as attribution indicators.

As previously mentioned, the LIME algorithm requires an image segmentation algorithm to work. Which segmentation algorithm is left to the users discretion. One downside mentioned in the LIME paper[27] is that sometimes image segments cannot explain why a specific prediction was made. Imagine trying to generate explanations for a classifier that is trained to detect sepia toned images. There are no specific segments that could be highlighted in an explanation. This simple example clearly illustrates a limitation of not just the LIME algorithm, but all feature attribution XAI methods.

4.5.5 LIME and Heading Regression

As was the case with Integrated Gradients, LIME must also be modified before it can perform feature attribution on heading regression results. However, the modifications made to the LIME based method are quite different from the ones made to the Integrated Gradients based method. This is

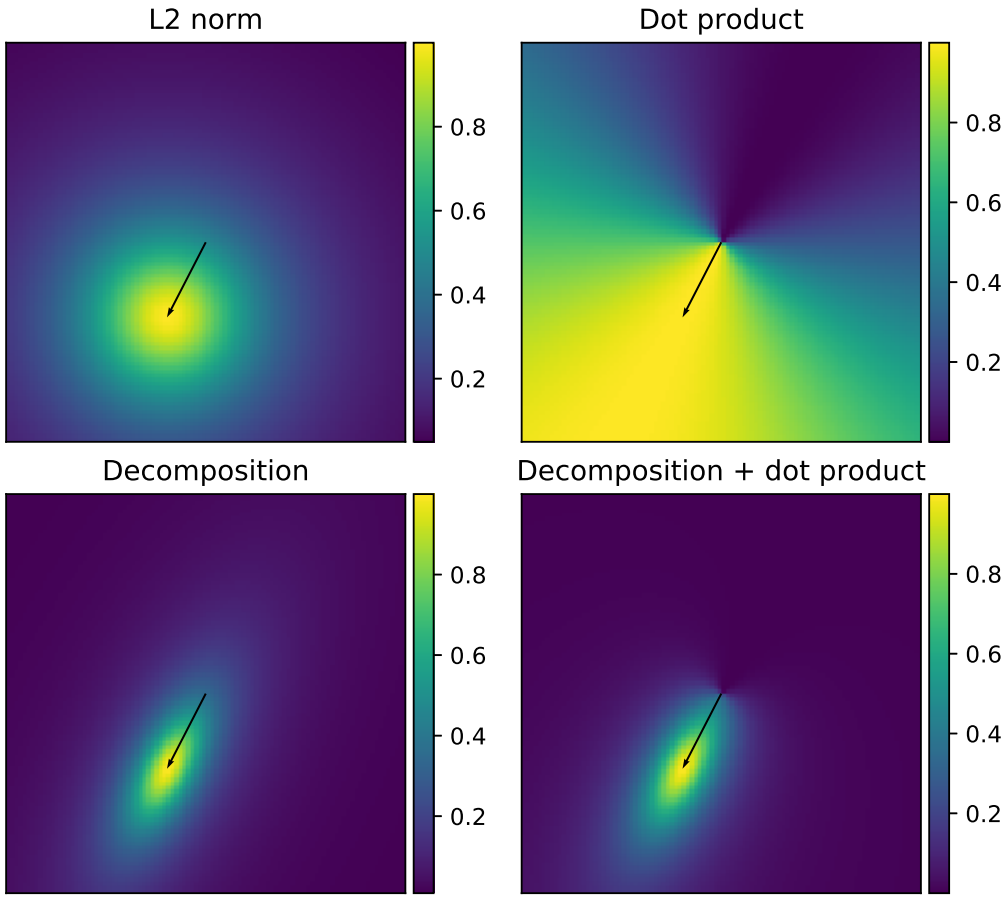


Figure 4.11: A few mapping functions that map from the originally predicted heading, represented by the black arrow, to the emulated classification scores.

because the LIME algorithm trains a surrogate model to emulate the primary model's classification predictions locally, its output is thus limited to between 0 and 1. Since the primary network outputs a two dimensional vector representing the predicted heading it must first be converted to a value between 0 and 1 before using it to train the surrogate model.

It is clear that there are many ways of doing this and that this thesis will only be able to explore a few of them. This thesis proposes a method that converts the heading vectors to a valid classification score by comparing it to the original heading prediction. Intuitively, if the output heading prediction vector stemming from the altered input image is close to the original, it should result in an emulated classification score close to 1. Similarly, if the output heading prediction vector is very different from the original, it should be assigned an emulated classification score close to 0. This means that one is free to experiment with a limitless selection of different mapping functions, as long as the function constrains the emulated classification score to between 0 and 1.

Figure 4.11 illustrates four different mapping functions. For easier notation, define the error vector, given by

$$\mathbf{h}_{err} = \mathbf{h}_{new} - \mathbf{h} \tag{4.7}$$

where $\mathbf{h}_{new}$ is the newly predicted heading vector, and $\mathbf{h}$ is the originally predicted heading vector. The first mapping function is based on simply measuring the euclidean distance, via the error vector, and mapping it to between 0 and 1 through

$$m(\mathbf{h}_{err}) = \frac{1}{1 + \alpha \left\| \mathbf{h}_{err} \right\|} \tag{4.8}$$

where m is the mapping function, α is a parameter which determines how harshly deviations are punished, and $\mathbf{h}_{err}$ is the error vector as defined in (4.7). At first glance this method might seem fine to use, but it has some downsides stemming from how the final heading angle is predicted. Since the final predicted heading angle is generated by the atan2 function, only the direction of the heading vector matters, not its magnitude. Imagine that the network originally predicts a heading vector to $(-0.5, -0.5)$, indicating a vessel heading of $225°$, somewhat similar to the one shown in Figure 4.11. Now imagine that while training the surrogate model, two altered versions of the input image produce the predicted headings $(-1.2, -1.2)$ and $(0.2, 0.2)$. Using the mapping function defined in Equation 4.8 both of these vectors will result in equal classification scores, since are the same euclidean distance from the originally predicted vector. Even though the first vector reproduces the initially predicted heading angle exactly, while the second vector points in the opposite direction, leading to a predicted heading of $45°$! Clearly these predictions should not be scored the same. The next method tries to alleviate this issue.

The next mapping function is based upon the dot product between the original heading vector and the new heading vector. This allows it to correctly punish heading vectors that point in different directions than the original prediction. In the example provided in the previous paragraph the second heading vector would now receive an emulated classification score of 0, since it points in the opposite direction than the originally predicted heading vector. The mapping is shown in the upper-right sub-figure in Figure 4.11 and can be expressed as

$$m(\mathbf{h}, \mathbf{h}_{new}) = \frac{\frac{\mathbf{h}}{\|\mathbf{h}\|} \cdot \frac{\mathbf{h}_{new}}{\|\mathbf{h}_{new}\|}}{2} + \frac{1}{2}$$

where m is the mapping function, $\mathbf{h}_{new}$ is the newly predicted heading vector, and $\mathbf{h}$ is the originally predicted heading vector. This method suffers from another problem though. A predicted heading vector of small magnitude can indicate that the detected object has weak directionality, since non-directional classes' heading vectors are defined as the zero vector. Therefore, the mapping function should punish changes to the predicted heading vector magnitude.

The two next mapping functions introduce improvements over the previous methods. These decompose the error vector into parallel and orthogonal components with regards to the originally predicted heading vector. This lets them punish orthogonal deviations harder than parallel deviations, which might be good, since only orthogonal deviations actually change the final predicted heading. The bottom-left sub-figure in Figure 4.11 illustrates this mapping function. It can be written as

$$m(\mathbf{h}, \mathbf{h}_{err}) = \frac{1}{1 + \left(\alpha \left\| \frac{\mathbf{h} \cdot \mathbf{h}_{err}}{\|\mathbf{h}\|} \frac{\mathbf{h}}{\|\mathbf{h}\|} \right\| + \beta \left\| \mathbf{h}_{err} - \frac{\mathbf{h} \cdot \mathbf{h}_{err}}{\|\mathbf{h}\|} \frac{\mathbf{h}}{\|\mathbf{h}\|} \right\|\right)} \tag{4.9}$$

where m is the mapping function, α and β are parameters that decide how harshly parallel and orthogonal deviations are punished, $\mathbf{h}_{err}$ is the error vector, and $\mathbf{h}$ is the originally predicted heading vector. However, this method suffers from the same issue as the L2 norm based method discussed earlier, and might not punish vectors that point in very different direction as harshly as they should be. The final mapping functions, shown in the bottom-right sub-figure in Figure 4.8, solves this by multiplying the mapping function defined in Equation 4.9 with the dot product of $\mathbf{h}$ and $\mathbf{h}_{new}$. This can be expressed as

$$m(\mathbf{h}, \mathbf{h}_{new}, \mathbf{h}_{err}) = \frac{\frac{\mathbf{h}}{\|\mathbf{h}\|} \cdot \frac{\mathbf{h}_{new}}{\|\mathbf{h}_{new}\|}}{1 + \left(\alpha \left\| \frac{\mathbf{h} \cdot \mathbf{h}_{err}}{\|\mathbf{h}\|} \frac{\mathbf{h}}{\|\mathbf{h}\|} \right\| + \beta \left\| \mathbf{h}_{err} - \frac{\mathbf{h} \cdot \mathbf{h}_{err}}{\|\mathbf{h}\|} \frac{\mathbf{h}}{\|\mathbf{h}\|} \right\|\right)} \tag{4.10}$$

and theoretically offers 'the best of both' with regards to offering a mapping that punishes heading angle deviations, parallel deviations, and orthogonal deviations - all while being configurable by tuning α and β.

4.5.6 LIME and Cropped Analysis

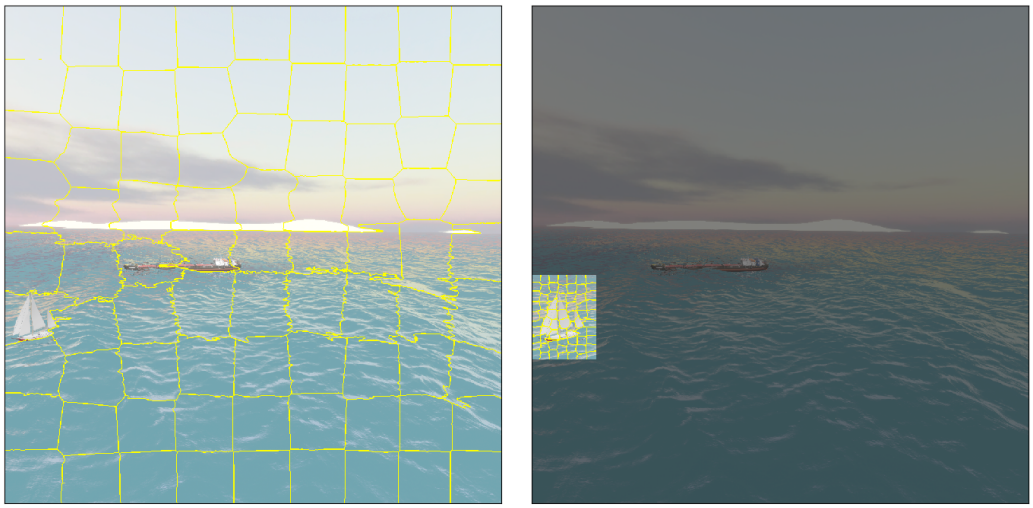


Figure 4.12: An example of the cropped analysis segmentation mode. The original segmentation is to the left, while the cropped analysis segmentation is to the right. Areas inaccessible to LIME are darkened. The segments are highlighted in yellow. The target vessel is the sailboat.

In the project thesis preceding this thesis, LIME sometimes struggled to produce good feature attributions for detected objects, especially when the detected object appeared small in the image

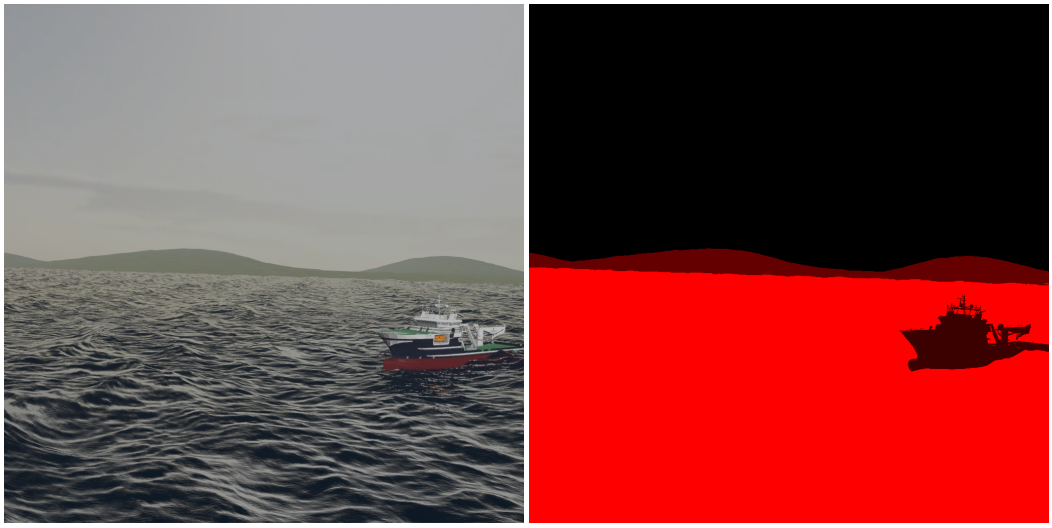
frame. This issue was caused by the image segmentation algorithm employed by LIME, which always segmented the entire image, even when the target object only occupied a small portion of it. Which specific segmentation method used did not matter; the issue was that it always segmented the entire image. This often meant that the object of interest was only covered by a few, very large segments, which reduced the fidelity of the feature attributions. This unfortunate segmentation led to another issue. It made it more difficult for the surrogate model to emulate the primary model, since the number of segments had to be very large in order to produce high fidelity feature attributions describing the object. This meant that the surrogate model had to emulate the main model over more dimensions, which was suspected to degrade the quality of the feature attributions.

This thesis tries to alleviate this issue by introducing LIME with cropped analysis. This is a simple modification that reformulates the LIME algorithm to only work within an area close to the predicted bounding box around the target object. This allows high fidelity segmentation around the target object, while keeping the overall number of segmentations low. This method comes with a trade-off, it being that the LIME algorithm cannot detect influences stemming from outside the area around the predicted object bounding box. The reader might remember the example shown in Figure 4.2 where the LIME method determined that the classifier used the snow on the ground instead of the features of the dog/wolf as the basis for its predictions. This would not be possible with this solution, since the snow would likely be outside the canine's bounding box and therefore be inaccessible to the LIME algorithm. Figure 4.12 shows an example of the new cropped analysis mode. It is clear that the cropped analysis segmentation offers much higher fidelity segmentation of the target object.

Chapter 5

Synthetic Training Data Acquisition

This chapter presents the program that generates the synthetic datasets using the Kongsberg Cogs graphics engine. It also covers what visual effects the generated datasets can contain and how they can be configured. Additionally, it discusses how the information constituting the dataset samples is collected, how the procedural terrain generator works, and how small objects are handled in the program. It also discusses other information that can be extracted from the 3D scenes, such as a pixel-perfect depth-map and the relative headings of the objects present within the scene.



(a) A scene generated by the Cogs 3D rendering engine. **(b)** The accompanying object masks.

Figure 5.1: Output from the program for synthetic dataset generation.

5.1 Kongsberg Cogs

Cogs is an in-house 3D rendering-engine developed by Kongsberg Digital, a company within the Kongsberg group which specializes in supplying software and digital solutions to the maritime, oil and gas, and utility sectors. It is designed to be a lightweight and flexible 3D graphics renderer for use in their projects. For Kongsberg Digital it provides vertical integration, meaning that the team at Kongsberg Digital has full control over every part of the graphical systems and can guarantee its operation. It is designed to be flexible and modular, making it easy to modify for special projects, such as generating synthetic training data for machine learning applications. In this project Kongsberg Cogs was used as the 3D graphics engine for rendering the synthetic images using the 3D models and other parameters supplied from the rest of the synthetic dataset generation software.

5.2 Synthetic Data Generation in Practice

Fortunately, the Cogs engine has already been used for data acquisition similar to what is required for this thesis. The framework for mask generation is already implemented and easy to use. For increased modularity it allows configuration via a Python API. This API supports asynchronous communication between a Python script and the Cogs engine over a local internet connection. Through this bridge Cogs sends the rendered images to the rest of the synthetic dataset generation program for storage. During the thesis this bridge has been expanded as needed, thanks to continued support from contacts at Kongsberg Digital.

The Cogs graphics engine supports dynamic loading and removal of 3D models from the scenes.

These can then be placed, scaled, and rotated as desired. Cogs also includes an ocean simulation with configurable wave-height, wave-period, and color. This enables a wide variety of sea states to be generated. Other elements of the simulation that can be varied between the samples are: The fog strength and color, the terrain shape and color, the sun location, the camera rotation, the camera height, the camera field of view, and the appearance of the sky. Each of these parameters can be sampled from a chosen probability distribution with only small changes to the code. The parameters describing the sampling probability distributions are defined in a configuration class which is passed into the program at the start of the synthetic dataset generation. During the development of the system multiple combinations of parameters were experimented with.

5.3 The Architecture for Dataset Generation

The Python program that interfaces with Cogs to produce a synthetic dataset is split into three modules: the Scene-Populator, the Scene-Randomizer, and the Sample-Generator. The architecture is shown in Figure 5.2. The three modules communicate with the Cogs program through a separate Python bridge, and are responsible for various parts of the sample generation process.

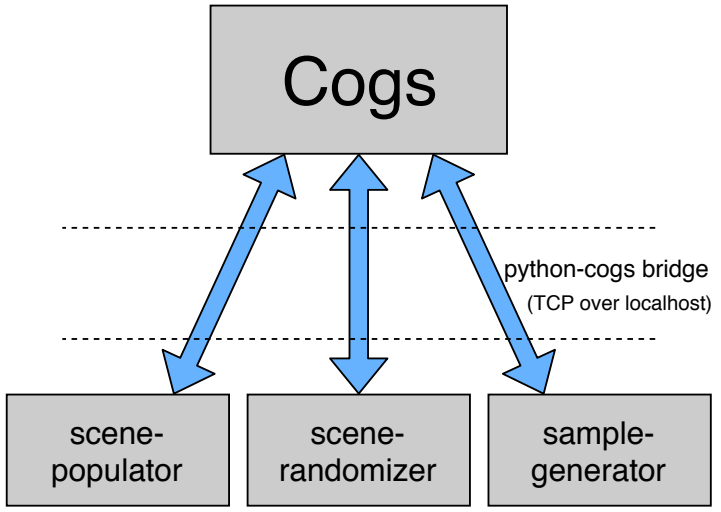


Figure 5.2: The three parts of the dataset generation program communicates with Cogs through the Python-Cogs bridge.

When generating a dataset, Cogs is initialized with an empty scene. The Scene-Populator then loads a given number of models into the scene. It randomly samples a class from a uniform distribution and then samples a 3D model representing that class. If it did not do it this way, instead directly sampling the 3D models, it could lead to biases when the number of models representing each class is not equal. For example: if it instead loaded the models with the naive method. Tank ships would occur 6 times more frequently than offshore wind turbines, because there are 6 times more 3D models representing tank ships. This could create an unbalanced dataset, which would cause the trained

model to be biased.

When the 3D models have been loaded into the Cogs scene, the Scene-Randomizer takes over. It is responsible for making each scene look different. First, it places the 3D models loaded by the Scene-Populator in a random position visible by the camera, taking the cameras field of view into account. It randomly rotates the 3D models and ensures that they do not overlap. It also measures the size of the 3D model and lifts it slightly to prevent flooding if the object is small. This is further discussed in Section 5.5. The Scene-Randomizer is also responsible for sampling all the other parameters discussed in the previous section, as well as generating the background terrain, which is discussed in Section 5.4.

After a scene has been prepared by the Scene-Populator and the Scene-Randomizer, the Sample-Generator downloads the scene image, scene segmentation, scene depth-map, and other information via its Python bridge. It then saves the scene image and scene segmentation as 24 bit RGB .PNG files and the depth-map as a 16 bit b/w .PNG file. All of the information about the objects present in the scene is stored in a separate .JSON text file, this file includes information such as the relative heading of the 3D models in the scene as well as their distances from the camera. The Sample-Generator is also responsible for keeping track of the complete dataset and logs how many times the different classes appear. In the end, a finished dataset sample consists of four files: the scene image, the scene segmentation image, the depth-map image, and the .JSON file containing various information. When all four files have been saved to disk the program resets and the next sample is generated, starting with the Scene-Populator.

5.4 Terrain Generation

Cogs has a built-in terrain renderer. It requires three textures to operate: a height-map, a color-map, and a normal-map. The height-map describes the height of the terrain at a given point in the scene, the color-map describes the color at a given point, and the normal-map increases the graphical fidelity of the terrain by encoding the surface normal as an RGB image. Normal-maps are discussed in Section 2.4.3.

The height-map is generated through a combination of Perlin noise, Gaussian noise, and Gaussian filtering. Perlin noise is used because of its ability to create continuous noise patterns. A naive implementation might use uniformly sampled noise. Unfortunately this solution would yield discontinuous, unnatural looking terrain. This is further discussed in Section 2.4.2. The initial Perlin noise is then filtered to further smooth it. A high frequency, low amplitude Gaussian noise is then applied to create realistic looking formations along the waterline. The terrain generation defines three types of terrain: land, islands, and ocean. Each of these have varying noise and filtering parameters to generate the desired look of the terrain. There is no set way of finding the correct values for these parameters and in this project they were found by trial and error.

There are three types terrain colors defined in the terrain generator. These are: a green color representing grass, a white color representing snow, and a gray color representing bedrock. The white and gray terrain types are randomly sampled within specified ranges and combined with Gaussian

(a) The color-map generated by the terrain generation algorithm. The terrain generated has a green/brown color.

(b) The height-map generated by the terrain generation. The dark areas have low elevation, while the brighter areas correspond to higher elevation. This formation forms islands in the distance.

(c) The normal-map generated by the terrain generation.

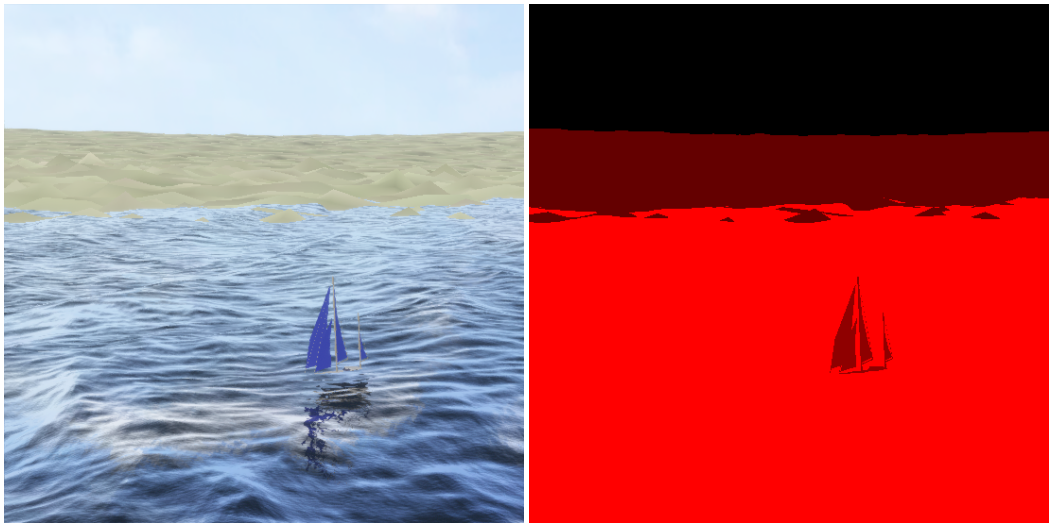
Figure 5.3: The textures used when generating the image sample shown in Figure 5.1a and 5.1b.

noise to induce variations within the color. The green terrain types are sampled in the HSV colorspace, which makes it easier to sample natural looking hues. The HSV colorspace is further discussed in Section 2.4.4. During the image sample generation, the terrain type is selected through sampling a configurable probability distribution.

5.5 Handling Small Boats

There are about 150 different 3D models used in the dataset generation. These 3D models range from cruise ships, to offshore wind turbines, to small leisure boats. This means that there can be a large discrepancy in scale between the objects placed within a scene. During testing it was found that parameters that created natural looking images of cruise ships would not create usable images of small motor boats. To elaborate: it is reasonable to expect the object detection system to detect and classify a 250m container ship at a distance of 2 kilometers, but to expect the same for a small leisure boat is not realistic. Especially not if the significant wave height is 12 meters. Naturally, the discussion above would be different when simulating a small field of view camera (a camera with high zoom). This means that the smaller vessels should be placed closer to the camera in order to create the best possible training data for machine learning models.

Another problem stemmed from the lack of buoyancy simulation in the Cogs ocean simulation. In the real world a boat is expected to float on top of the water, but in Cogs objects have fixed position in 3D space which will not change due to buoyancy forces. This is not a problem for large ships such as oil tankers, since their waterline changes several meters due to different load conditions anyway. For small boats however, this could cause the entire hull to be submerged under water. Early in the development of the synthetic dataset generation system this led to a bias in the datasets especially visible for the sailboat class. Because of their small size the model had never seen a non-flooded



(a) The scene image. **(b)** The object masks.

Figure 5.4: A partially submerged sailboat. This issue caused biases early in the development of the system.

sailboat and when it encountered real world images of sailboats with their hulls visible it classified them incorrectly.

To alleviate both problems described in this section, the Scene-Randomizer was modified to measure the size of the 3D models it placed. This measurement influences two things. Firstly, it limits how far small boats can be placed from the camera by enforcing a maximum distance according to

$$d_{max} = \frac{min(W_{3d-model}, H_{3d-model}, D_{3d-model})}{tan(fov * p_{min}/res)} \tag{5.1}$$

where d_{max} is the largest allowable distance that the object can be placed from the camera, $W_{3d-model}$, $H_{3d-model}$, $D_{3d-model}$ are the width, height, and depth of the 3D model in meters, p_{min} is the smallest cross section allowed in pixels, fov is the camera field of view in radians, and res is the resolution in the produced images. This distance limit ensured that all the vessels in the images were visible enough that the machine learning systems should be able to detect them. To alleviate the flooded-hull problem, the system was modified to also lift the 3D models higher in the water by translating them in the Z-direction if they were under a threshold size. This led to fewer small vessels being partially underwater in the generated images, which hopefully reduced the effect of the aforementioned bias, and improved the model's performance.

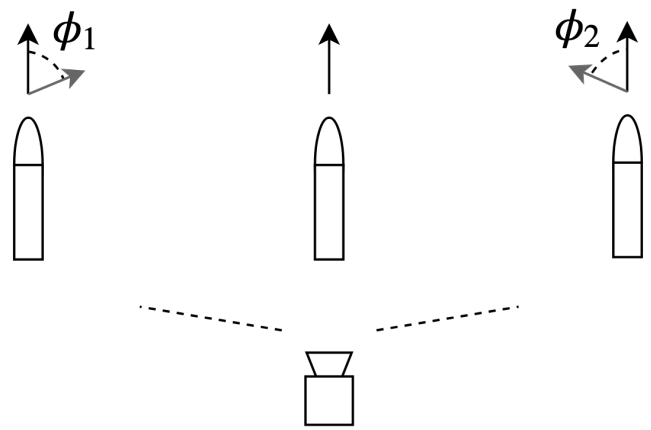


Figure 5.5: The true vessel headings and the apparent heading from the perspective of the camera. The true headings are shown in black and the apparent headings are shown in grey.

5.6 The Vessel's Position's Effect on the Apparent Heading

During the development of the synthetic dataset generation a concern arose. Perhaps the detected vessel's horizontal position in the image frame could impact the performance of the heading prediction system, since the vessel's apparent heading would change? This concern led to the experiment discussed in Section 6.2.2 and 7.1.2 which tests this hypothesis. To facilitate this experiment, a separate measurement of the vessel's apparent heading was added. The apparent heading measurement includes an offset which compensates for the object's horizontal position in the image frame. Figure 5.5 highlights this effect by showing the true vessel headings as well as the apparent headings caused by the perspective of the camera. This effect is most pronounced when the vessel's position is towards the horizontal edges of the image and is strengthened when the image has a large field of view. One can observe this effect by closing one eye, placing ones finger in front of the open eye while pointing it towards it. Then, move the finger towards the side of the head without rotating the finger. You should see the apparent heading change even though the true heading does not. Naturally, it is important that you do not rotate your finger while doing this, otherwise the finger's true heading would change as well. The offset is simply calculated by using the 3D model's position in the scene. The offset is given by

$$\phi = atan(\frac{pos_y}{pos_x}) \tag{5.2}$$

with pos_x being the vessel's distance from the camera parallel with the camera view and pos_y being the distance from the camera orthogonal to the camera view. The apparent vessel heading is then found by

$$heading_{apparent} = heading_{true} - \phi \tag{5.3}$$

with $heading_{true}$ representing the 3D model's true heading in the 3D scene. Finally, the heading is normalized to between $0°/360°$. To recover the true heading from a heading predicted by a model

trained with this offset, the following formula is used

$$heading_{true} = heading_{apparent} + \frac{px_x}{res_x} fov \tag{5.4}$$

where px_x is the horizontal distance of the detected object from the image centerline in pixels, res_x is the horizontal resolution of the image, and fov is the field of view of the image. Again, it is important to remember to normalize the result to between $0°/360°$. These transformations allow the heading estimation systems to be trained to predict both true and apparent heading.

So why does it matter whether the heading estimation system predicts the true heading or the apparent heading? After all, a human could easily consider the relative position of a vessel in his/hers estimations when asked to predict a vessel's heading. However, because the Mask RCNN architecture does not include feedback from the position of the detected objects in the image frame to the heading estimation module it is unable to do this. Therefore the system might be less accurate when tasked with predicting the true heading directly; it might perform better if it first predicts the apparent heading and then transforms it to the true heading through Equation 5.4. The experiment discussed in Section 6.2.2 and 7.1.2 explores whether one option performs better than the other.

Chapter 6

Experiments

This chapter presents the three experiments performed in this thesis. The chapter starts with discussing the synthetic maritime datasets used and the experimental setup on which the machine learning systems were trained. Then, the heading estimation experiment is discussed. In it, three modified versions of the Mask RCNN architecture are trained to estimate the heading of detected objects. The three models are compared in depth.

The second experiment to be discussed is the feature attribution experiment. It involves modifying two feature attribution methods to generate feature attributions for the heading predictions made by one of the modified versions of the Mask RCNN networks. These feature attribution methods are first validated on a simple two dimensional toy dataset, before they are used to gain insight into the Mask RCNN model trained on the synthetic maritime dataset.

The third experiment evaluates methods for including depth information in the Mask RCNN model's input. Two methods are compared. One which simply adds the depth information as an additional image channel and one which gradually merges the depth information into the Mask RCNN's feature extractor network. Both these methods are compared to a baseline model without access to depth information.

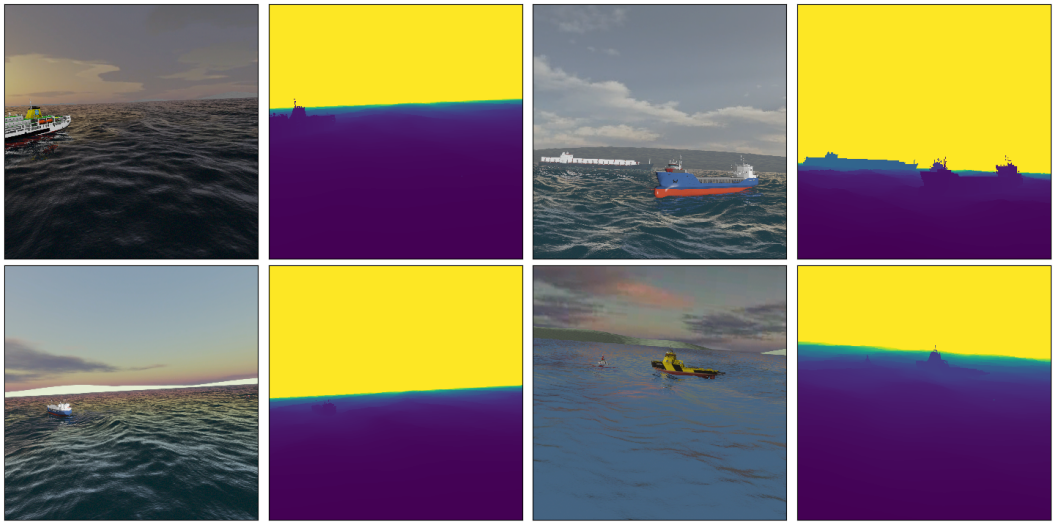


Figure 6.1: Some samples from the synthetic dataset with accompanying depth-maps.

6.1 The Dataset and Experimental Setup

For use in this thesis' experiments, a synthetic dataset consisting of 22000 images was created with the program discussed in Chapter 5. The dataset was split into two parts, a training dataset made up of 20000 samples, and a validation dataset consisting of 2000 samples. The dataset created utilized $\sim$ 150 unique 3D models of ships, boats, and offshore wind turbines. The dataset's defined vessel classes and the number of 3D models describing each class are shown in Table 6.1. The models were sampled according to the method described in Section 5.3 in order to avoid class imbalances. The generated images had a resolution of 1024 by 1024 pixels and the final dataset occupied 56.4 GB of disk space. Each sample consisted of four files: the scene image, the instance segmentation masks, the depth-map, and the info file.

Class	# models
Cargo ship	14
Container ship	13
Emergency ship	15
Passenger ship	21
Tanker ship	35
Tug boat	3
Utility ship	10
Wind turbine	6
Motor yacht	25
Sail yacht	8
Terrain	n/a
Sky	n/a
Free space (sea)	n/a

Table 6.1: The classes generated with the Cogs synthetic dataset generation program.

The models used in this thesis were trained on a workstation provided by Kongsberg Digital. It was equipped with a NVIDIA 1080ti graphics card, as well as an Intel i7-7700k CPU and 32 GB 2133 MHz RAM. The workstation also performed the feature attribution experiments and was shared with another student.

6.2 Heading Estimation

All the Mask RCNN models that took part in this experiment were each trained for a total of 1000000 training steps with a stepped learning rate shown in Table 6.2, this took roughly 22.5 hours per model. To prevent overfitting, the final models used in the experiments were the ones that achieved the lowest loss function on the validation dataset during the training process. The models often achieved this between training step 800K-1000K. The dataset and hardware setup are discussed in Section 6.1. This experiment is based upon Matterport's Mask RCNN implementation[2] which is available open source on Github. The models were configured to use the ResNet101 feature extractor and to expect square images of 1024 by 1024 pixels. A learning momentum of 0.9 was used throughout the training process. The initial model weights had been pretrained on the COCO dataset[62] by Matterport. The networks were trained with mini-batch gradient descent with a batch size of 2. The batch normalization layers were frozen during the training process.

Training steps	Learning rate
0-300K	0.001
300K-600K	0.0005
600K-800K	0.00025
800K-1000K	0.000125

Table 6.2: The learning rates used while training the Mask RCNN networks.

During the training phase the respective loss functions for the heading estimation, which are discussed in Section 3.2.1 and 3.2.2, were simply added to the existing loss function for the entire Mask RCNN architecture. This meant that the networks were trained with the composite loss function

$$L_{mrcnn} = L_{rpn\,class} + L_{rpn\,bbox} + L_{class} + L_{bbox} + L_{mask} + L_{heading} \tag{6.1}$$

where $L_{rpn\,class}$ is the loss function describing the RPNs' classification performance (cross-entropy), $L_{rpn\,bbox}$ is the RPN's bounding box loss function (smooth L1), L_{class} is the loss function for the final classification (cross-entropy), L_{bbox} is the final bounding box loss function (smooth L1), L_{mask} is the final mask loss function (pixel-wise binary cross-entropy), and $L_{heading}$ is the loss function for the chosen heading estimation method. $L_{heading}$ can be configured as either smooth L1 loss or categorical cross-entropy loss, depending on whether the network is configured to estimate headings through a unit vector approach or a classification approach. It is clear from this formulation, that it would be possible to multiply the heading estimation loss function with a constant to either increase or decrease its impact on the composite loss function, which in turn would impact how much the pretrained network would change to accommodate added heading estimation task. This was however, not done during the experiments.

An important characteristic of computer vision algorithms such as the ones used in this work is their inference time. Naturally, this would heavily impact how the algorithm could be applied in real world applications. Nonetheless, this thesis does not focus on measuring the models efficiency or inference time mainly because not much effort has been put into optimizing the methods implemented. Time has instead been spent on analyzing the methods and developing visualization to aid in this.

#	Heading estimation method	Heading measurement	Note
1	Single unit vector	True heading	
2	Single unit vector	Apparent heading	
3	N. unit vectors	True heading	
4	N. unit vectors	Apparent heading	
5	Classification	True heading	M = 7, N = 8
6	Classification	Apparent heading	M = 7, N = 8
7	-	-	Unmodified Mask RCNN

Table 6.3: The models trained in this experiment.

As noted in Section 3.2.2, M and N are important parameters for the classification-based heading estimation method. M and N describe how many distinct classification problems the problem is formulated as and how many discrete headings each of the classification problems contain, respectively. [25], the paper that first implemented the classification reformulation, experimented with various combinations of M and N. This thesis does not examine possible combinations of these two values in depth, instead using $M = 7$ and $N = 8$ which gives a total of 56 discrete heading values. Note that internally the Mask RCNN model actually estimates $N = 9$ values, but the first is reserved to represent the non-directional case and is removed before the mean shift algorithm.

As noted in Section 3.2.3, four of the thirteen defined classes are defined as directionless. This means that their headings are defined to be the zero vector, or the non-directional classification if the model is configured to use the classification-based heading estimation method. The following classes are defined as directionless: ocean, terrain, sky, and wind-turbines. Technically, wind-turbines could be defined to have a direction equal to the orientation of the turbine, but during development performance was found to improve if it was defined as directionless.

Most of the following experiments are based on the models' performances on the validation dataset, which consists of 2000 images. Each model was asked to generate predictions for all of the 2000 samples. These predictions, along with data about the vessel instances in the images, were logged and used as a basis for the following analysis. In total 7 models were trained during this experiment, an overview of these is shown in Table 6.3. If not otherwise stated, the models used in the experiments are configured to predict true heading, not apparent heading.

6.2.1 Prediction Accuracy on a Class by Class Basis

The various measurements for heading estimation accuracy discussed in Section 3.3 will be applied to the three different models' results. These will shed light on some basic aspects of their performances. Measurements such as the meanAE, the medianAE, and other accuracy measurements can together provide a comprehensive overview of the models' performances. Since the Mask RCNN models are trained to classify a wide variety of defined classes, the inter-class heading estimation performance is also of interest. To evaluate it, the models' prediction accuracies are logged on a per class basis. This class specific information is then used to create angular error histograms and other measurements per class. The defined classes are shown in Table 6.1.

6.2.2 Apparent Heading vs. True Heading

As discussed in Section 5.6 a detected vessel's horizontal position in the image frame influences its apparent heading. It would be interesting to check whether the performance of the heading estimation systems would increase if the systems were tasked with estimating the apparent heading instead of the true heading. To test this, each model trained in this experiment has an identical twin, which was trained to estimate the apparent heading instead of the true heading. By analyzing and comparing the performance differences between each set of models, one can find out if there are any benefits to estimating the apparent heading instead of the true heading.

6.2.3 Prediction Accuracy as a Function of Object Distance

As the distance to a detected vessel increases, its angular size as seen from the camera decreases. This decrease in angular size means that, if all other things are kept equal, the number of pixels representing that object decreases. This should make the task of estimating the object's heading more difficult, since the heading prediction sub-network is constrained to working with a lower fidelity input feature map. One should however note that other parameters also affect the angular size of an vessel, such as the camera's field of view, the orientation of the vessel, and the sea state. Still, the prediction accuracy vs. distance plots should provide an insight into whether the models' predictions are robust with regards to a detected vessel's distance.

If systems such as the ones in this thesis are implemented in real world operations, they would naturally need to be robust with respect to a detected vessel's distance. In a maritime scenario this could realistically be distances from tens of meters to several kilometers. It is thanks to synthetic dataset generation that this can be analysed at such detail. If one were working with a real world dataset it is unlikely that accurate measurements of the object distances would have been accessible. This is an example of how synthetic datasets can facilitate more detailed analysis of computer vision solutions.

6.2.4 Prediction Accuracy as a Function of Object Heading

As the heading of a detected vessel changes it impacts two things: Firstly, it changes the visible size of the object. A ship with its bow pointing towards the camera appears smaller than a ship showing its entire length. As noted in the previous section, this should make the task of estimating the object's heading more difficult, since the heading prediction sub-network is constrained to working with a lower fidelity input feature map. The second and maybe most important effect is that a change in heading changes the entire appearance of the vessel, as well as which of its features are visible from the camera's perspective. Note that since a roll motion can be applied to the camera between the samples, the vessel's heading is not the only changing factor. The camera's height above the water is also changed between samples, which might also impact the apparent heading of the detected vessels. However, the vessel's heading should be the dominant factor.

Creating computer vision algorithms that are robust with regards to rotation is one of the main challenges within the field. It is therefore of great interest to see how well the model performs as a function of the detected vessels' headings.

6.2.5 Comparison with Unmodified Mask RCNN Architecture

Since this thesis modifies the Mask RCNN architecture to also perform heading estimation, it is of interest to examine how the other aspects of the Mask RCNN's performance are impacted. In order to test this, another Mask RCNN model without the heading estimations modification is trained on the same dataset to serve as a baseline. Hopefully, the addition of heading prediction modules will not degrade the performances of the modified Mask RCNN models too much when compared to the baseline model.

6.2.6 Method Specific Evaluations

Some method specific aspects will also be examined. As a refresher, three different modifications to the Mask RCNN for estimating object's headings are compared in these experiments. The first two are based on estimating the vessel's heading as a vector pointing in the same direction as the detected vessel. The first method estimates one vector which is used for all classes, while the second method estimates one vector per defined class and samples the vector corresponding to the predicted class. One interesting target for analysis is the distribution of the class specific heading vectors produced by the second method. It would be interesting to see whether the network predicts similar heading vectors for each of the classes or if these are different. If they are similar, this could indicate that the network does not use class specific features in its predictions. To examine this, the class specific heading vectors generated for a set of samples will be visualized and analyzed.

Remember that the third heading estimation method reformulates the problem as a set of classification problems and regains the final heading vector through a clustering technique. The distribution of these classification scores before the clustering algorithm are of great interest. Maybe if the model is unsure about a heading prediction multiple peaks can be observed?

6.2.7 Comparison to Human Performance

In order to create a realistic benchmark for the models to be compared with, a survey of people's ability to estimate the heading of ships in images was performed. The survey was performed by distributing a simple program in which the user was presented with a set of samples from the validation dataset and asked to estimate the heading of one of the vessels in the image, indicated by a red bounding box. The program was written in Python using the Tkinter GUI module. A screenshot of the application is shown in Figure 6.2. The application is designed to be as simple as possible while allowing the user to input precise heading estimates, either numerically or by creating an arrow in the blue input box. When all the problems have been answered, the delivery button turns green and the user can deliver the predictions. This freezes the predictions and shows the ground truth headings

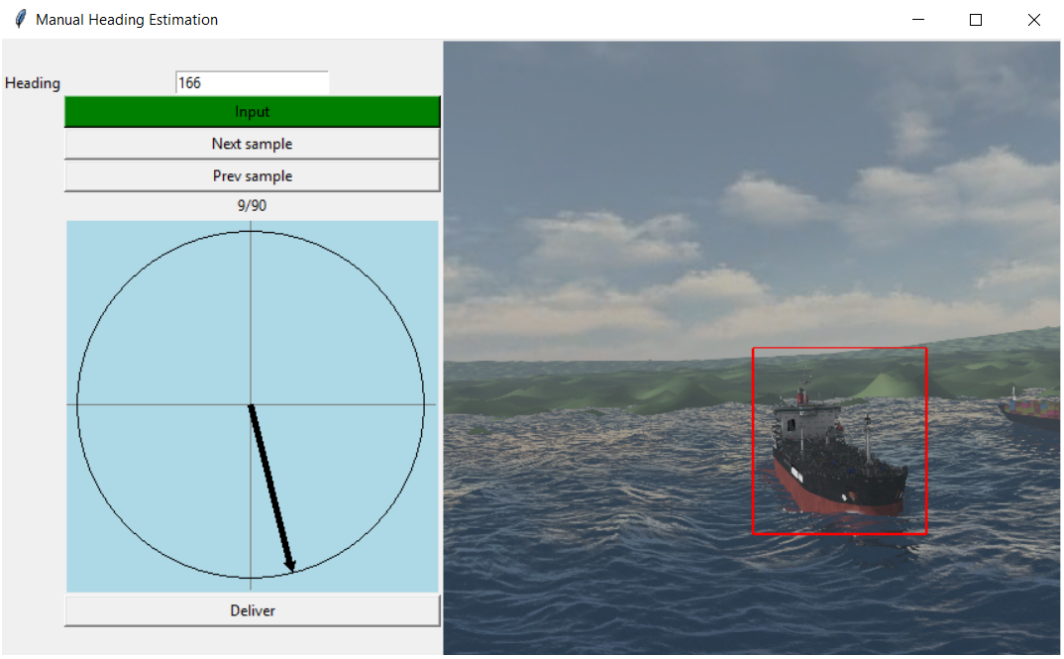
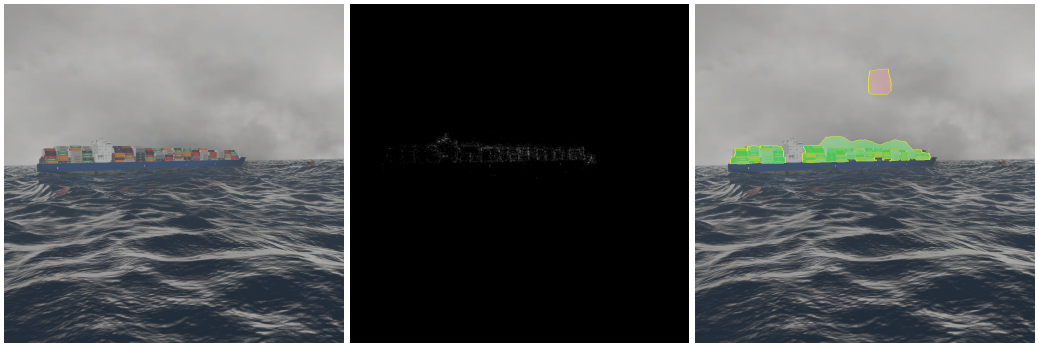


Figure 6.2: A screenshot from the application used in the survey of people's ability to estimate the heading of ships in images. Here the tanker inside the red box is the target.

as a green arrow.

The survey consisted of 90 samples of gradually increasing difficulty. The application kept track of each sample's difficulty as either 'easy', 'medium', or 'hard', to facilitate further analysis. Which difficulty a sample belonged to was decided manually during the creation of the testing application. The difficulties were evenly divided with 30 samples in each. The application also logged whether the target ship was partly occluded by either terrain, another vessel, or the edges of the image. To facilitate easy use, the application was 'compiled' into an .exe file with the Pyinstaller module. This way the user did not have to download Python and install the required modules to use the program.

In the end the survey had a total of 29 participants. The participants were mostly other students, friends of the author, and family members. It was also answered by a few maritime professionals, namely a maritime pilot, a captain and a deck officer. With each participant providing 90 samples, this gives a total sample size of 2610 samples, spread over a wide variety of difficulties. This should provide a good basis for comparisons with the methods implemented in this experiment. It is important to note that the participants only had one attempt at the survey, and that most participants would likely improve if given more attempts. The participants were only given one attempt because the survey was meant to be as simple and require as little time as possible. This way, many people could participate. The survey was only meant to give an indication of human performance, not examine it precisely. That could be a topic of a later study.



(a) The containership classification is the target of the explanations.

(b) The areas highlighted by the Integrated Gradients algorithm.

(c) The areas highlighted by the LIME algorithm.

Figure 6.3: Explanations generated by some feature attribution algorithms. Observe that both methods highlight the containers as an important feature of the class. The results are from [1].

6.3 Feature Attribution on Heading Estimation

[1], the project thesis preceding this work, experimented with three different methods for feature attributions: examining the Jacobian matrix, Integrated Gradients, and LIME. These methods were tasked with generating explanations for a set of classification predictions from a Mask RCNN model by highlighting what parts of the objects most influenced the classification result. Some of the resulting explanations are highlighted in Figure 6.3. A natural progression from this work is to adapt the XAI methods to generate explanations for the heading estimates generated by the methods discussed in the previous section. In fact, that is exactly what this experiment aims to do, with some exceptions. This experiment will not examine the Jacobian matrix, since for all cases seen in the project thesis it performed worse than Integrated Gradients. The project thesis also found that LIME was not able to generate satisfactory explanations when the objects were small in the image frame. This experiment employs a modified version of LIME in an attempt to mitigate this problem. The modifications to LIME and Integrated Gradients are discussed in Section 4.5.5, 4.5.3, and 4.5.6. Of the three methods for heading estimation tested in the previous experiment, only the method using the single unit vector will be analysed in this experiment.

To effectively use the XAI methods to generate explanations for the heading estimations the ROIs must be frozen. This was also done in the project thesis. The following explanation is from the project thesis, with some small modifications. As a reminder, the Mask RCNN architecture consists of three parts. The first part is the feature extractor, which converts the input image to several feature maps using the ResNet101 feature extractor. The ResNet architecture is described in Section 2.3.6. The second part of the architecture is the ROI (region of interest) network. It examines the feature maps from part one and extracts the areas most likely to contain objects. These areas are the ROIs. The third and final layer then examines the parts of the feature map noted as ROIs and predicts the class, bounding box, heading, and the mask for each.

Since the XAI methods used in the experiments described in Section 6.3 were originally designed for simple one object classifier networks, and the Mask RCNN algorithm is a complex architecture

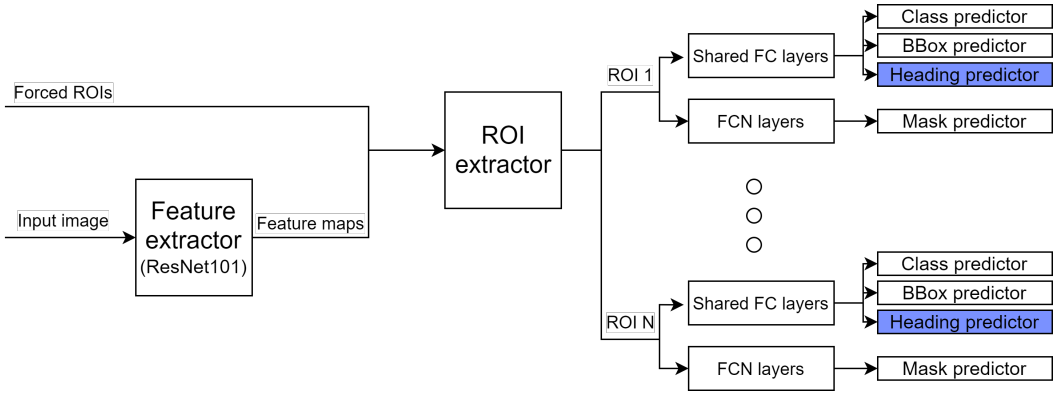


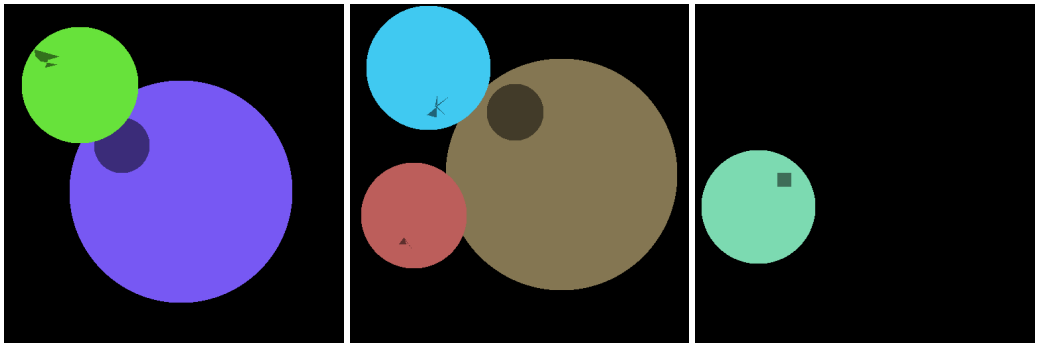
Figure 6.4: To use the XAI methods to generate explanations for the heading estimations the ROIs must be frozen. The heading classifier sub-networks are highlighted in blue. In this figure the ROI extractor represents the Region Proposal Network (RPN).

consisting of several sub-networks working together, some simplifications and assumptions had to be made in the experiments performed for this thesis. Internally, the RPN evaluates a set number of fixed anchors in the input feature map, computes the corresponding bounding box and predicts whether it contains a foreground object or a background object. This process of passing ROIs on to the final class prediction network is one of the defining features of RCNN networks and is explained further in Section 2.3.5. The XAI experiments performed in this thesis focuses on the heading prediction for a given instance. Therefore, the output from the RPN has been frozen in the experiments. This means that the network is forced to focus on the same areas of the feature map even if the input image changes. Normally changing the input image would cause different regions of the feature map to be highlighted by the RPN. Figure 6.4 shows the modified architecture performed in this thesis.

When using Integrated Gradients and LIME, which both alter the input image in some way while either calculating multiple gradients or classifications scores, an assumption is required. During normal operations, the final ROI containing a detected object might change as the input image changes. This will lead to a different crop of the feature map being used in the heading prediction module, and might lead to a different prediction result. The assumption that needs to be made when using Integrated Gradients and LIME with frozen ROIs is that this variability does not impact the XAI methods' abilities to generate explanations for the model predictions.

6.3.1 Validating the XAI Methods

It is hard to accurately evaluate the performance of XAI methods. This makes the development of XAI methods difficult, since in order to detect a performance improvement one needs to accurately measure it in the first place. So why is it difficult to evaluate the explanations generated for the heading estimations? To illustrate, think about how one would evaluate the attributions generated on the synthetic maritime dataset. A natural method would be to let the methods generate explanations for a number of predictions and see whether they are able to accurately highlight which features the



(a) (b) (c)

Figure 6.5: Three images from the ”Blob World” dataset.

model uses in its predictions. This would probably work great, if the features the heading prediction model bases its predictions on were known in advance. Unfortunately these are not known. This is why the XAI methods are developed in the first place. This section discusses what measures was made during the development of the XAI methods in order to overcome this fundamental problem.

It is now clear that evaluating the performance of the XAI methods developed in this thesis is difficult since we do not know which features the model bases its predictions on. During their development, the XAI methods were therefore tested on another dataset, in which the features that the model based its predictions on were known. This dataset was named ”Blob World” and as the name implies it was populated by circular blobs. These blobs come in different sizes and colors and lived on a black background. A handful example blobs are shown in Figure 6.5. A Mask RCNN model was then trained to detect these and predict their headings, which was defined by a geometric feature somewhere on the blob. The blobs are useful for evaluating the XAI explanations because the heading prediction system is forced to base its predictions on the location of the blobs’ geometric features relative to their bodies. The logic is as follows: if the trained Mask RCNN model is able to accurately predict the blobs’ headings it must have based its predictions on the geometric features since they are the only things that define the blobs’ headings. This means that we know which features the model bases its predictions on and are therefore able to accurately validate the accuracy of the XAI explanations.

So evaluating the explanations on the ”Blob World” dataset enables accurate implementations of the feature attribution methods. In the case of the method based on Integrated Gradients there are not that many parameters to tune, only the number of steps used when approximating the integral from Equation 4.1 as the sum shown in Equation 4.2. The synthetic dataset also allows comparisons between different merging functions. The modified Integrated Gradients method is further discussed in Section 4.5.3. LIME on the other hand, has several parameters and allows substantial tuning. Amongst other things it allows the user to tune the segmentation algorithm, the feature selection used while training the surrogate model, and the number of altered samples generated. The modifications to the LIME method is discussed in Section 4.5.5. Unfortunately, it is not known whether a model tuned to work on the blob dataset will be generalizable to other dataset.

6.3.2 Applying the XAI Methods to the Synthetic Maritime Dataset

The LIME based feature attribution method allows extensive tuning. To some extend the Integrated Gradients based method does as well. For this experiment the feature attribution configurations will be the ones that achieved the highest performance on the blob dataset discussed in the previous section. For more details around these configurations, see Section 7.2.1. The layout of the experiment will be similar to the one in [1]. Since accurate evaluation of feature attribution methods like the ones developed in this work is largely subjective, the discussion will focus on the different aspects of each method and what was learned during the experiment, not on concrete performance measurements.

The are two things of interest in this experiment. For one, it will be interesting to see whether the feature attribution models validated on the simple blob world dataset are able to generalize to another dataset, or if the models fail to do so. Secondly, if the feature attribution systems work as desired, it would be possible to gain insight into which features the Mask RCNN model bases its heading predictions on. Does the trained model use high level features such as the superstructure or the bow, or does it rely primarily on low level features such as the edges of the ship or simple textures? Since the heading prediction module is connected to the feature maps also used in the final classifier and bounding box regression, one would expect the feature attributions to primarily contain high level features. More details around how the Mask RCNN architecture has been modified to perform heading estimation are presented in Section 3.2.3. It would also be interesting to see how much the network relies on class specific features such as containers or sails in its predictions. Since only the single class unit vector approach will be analysed it is reasonable to expect that the network will be unable to rely much on class specific features in its predictions, since the single sub-network has been trained to estimate a single heading vector for all defined classes. It would also be interesting to see whether the Mask RCNN model uses the same features even if the target vessel's orientation changes. Does it use the same ship features when the ship is perpendicular to the camera as when it is parallel with the camera? Finally, it would be interesting to see whether the feature attribution models are able to highlight biases in the model, such as the famous husky-wolf example discussed in Section 4.1.

6.4 Including Pixel-Accurate Depth Information

The third and final experiment presented in this thesis is a bit shorter than the other two and serves two purposes: Firstly, it demonstrates how the modifications made to the synthetic dataset generation system in this thesis makes new experiments possible. The synthetic dataset generation system initially began development in the project thesis. With help from employees at Kongsberg Digital the synthetic dataset generation system is now able to extract pixel-accurate depth-maps from the Cogs graphics engine's internal 3D scenes. In the future, these depth-maps can be used to simulate various sensor systems such as LIDARs or RADARs, but in this experiment the raw depth-maps are used. The second purpose of this experiment is to examine various methods for fusing depth information into the feature maps in the feature extraction module of the Mask RCNN architecture.

As previously discussed in Section 2.5, different sensor systems have different strengths and weaknesses. A system which is able to combine sensor systems in such a such way that their combined weaknesses are reduced while maintaining their strengths is likely to achieve higher performance. In this experiment the depth information from the depth-maps will be combined with the RGB image of the scene. As their name implies, the depth-maps are rich in depth information, but are unable to provide the same contextual information as the RGB image. Hopefully, the final networks will learn to combine the two sensor systems and achieve higher performance than would otherwise be possible.

Two architectures for merging depth information into the Mask RCNN networks will be analysed. The first simply adds a fourth image channel in the input image representing the depth-map. This method requires little in the way of modifications to the network, but might yield lower performance as the Mask RCNN architecture was not originally designed for this and because of the potential mismatch in semantic value between the RGB image and the depth-map as suggested in [21].

The second architecture is a bit more involved. It is inspired by the architecture for fusing RADAR data and RGB images proposed in [21]. See Figure 2.25 for an illustration of this original architecture. Note that there are some differences between the architecture proposed in [21] and the one proposed in this thesis. For one, the original architecture downsampled the RADAR data through max pooling layers, while the architecture proposed in this work does it through average pooling. This is because the architecture proposed in this thesis works with full depth-maps instead of sparse RADAR measurements like the original architecture, where a positive value represented a RADAR return and downsampling through averaging would give a non-intuitive result. Another difference is that the original architecture concatenated the downsampled RADAR data and the feature maps in the feature pyramid, while the architecture proposed in this thesis combines them through addition after a 1x1 convolutional filter has been applied. This was done because avoiding the concatenation operation allows the use of pretrained weights, as the dimensions of the feature maps are not changed. This drastically reduces the time required for training, which allows quicker prototyping. The 1x1 convolutional filters use the RELU activation function. The original architecture is based upon RetinaNet and a VGG backbone[21], while the one presented in this work uses a ResNet-101-FPN as backbone. Figure 6.6 shows the architecture proposed in this thesis.

In addition to the two aforementioned models, a baseline model without the depth inclusion modifications, will also be trained. The baseline model allows us to see whether the two other models actually achieve better performance than they otherwise would have, had they not had access to the scene's depth information. All three models tested in this experiment use the classification reformulation method for heading prediction, configured with $M = 7$ and $N = 8$, and are configured to estimate the true heading of objects. Some important things must be noted. As described earlier in the text, all three models in this experiment utilize pretrained network parameters. Since the two architectures that utilize depth information include the information as a fourth image channel, the dimensionality of the first convolutional layer in the feature extractor is different than it otherwise would have been. This means that the networks cannot use the pretrained weights for this first layer. For consistency, the baseline model was also prevented from using these pretrained weights. Finally, the depth-maps concatenated to the input RGB image were normalized to floating point values be-

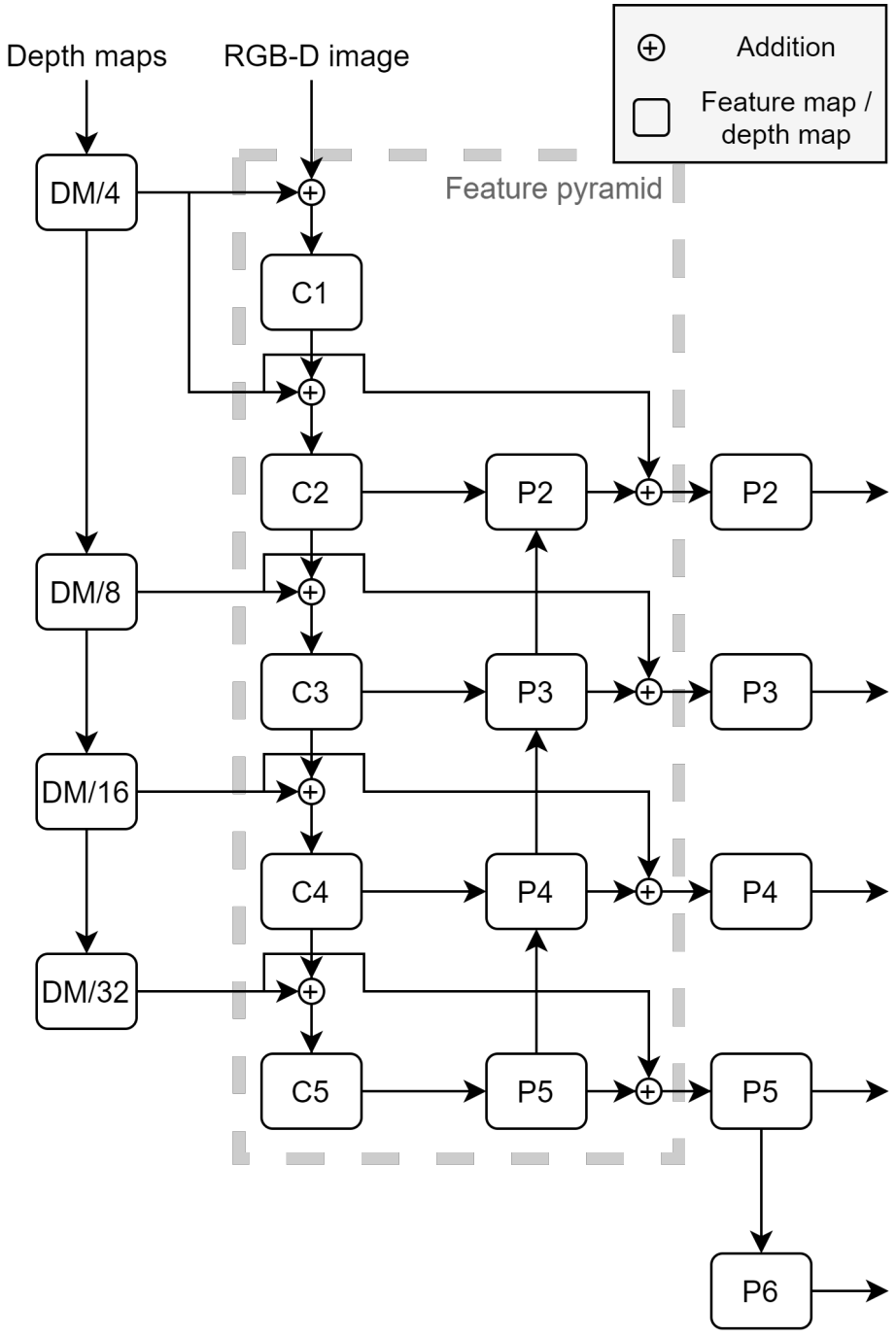


Figure 6.6: The proposed architecture for fusing depth-maps with images internally in the Mask RCNN's feature extractor network. This figure only shows the feature maps and the depth-maps for increased readability.

tween 0-255. This matches the value range of the RGB pixels and increased the stability during training.

The three models will be compared with respect to their validation loss during training, their classification performance measured through mAP, and their heading prediction performance. In the discussion, the model without access to depth information will be referred to as the baseline model, the one which adds the depth-maps as a fourth image channel will be called the simple-fuse model, and the model utilizing the more complex learnable fusion approach will be referred to as the deep-fuse model.

Chapter 7

Results and Discussion

This chapter discusses the results of the three experiments conducted in this thesis. The three methods for heading prediction are compared and analysed in detail. The two modified feature attribution methods are first validated on the 'Blob World' toy dataset, before used they are used to gain insight into the heading prediction system trained on the synthetic maritime dataset. Finally, the result from the third and final experiment is presented. It explored how the performance of the Mask RCNN architecture would change if the model had access to depth information.

Model	MeanAE	MedianAE	Acc22.5	Acc90
Single unit vector	41.25°	21.22°	51.96 %	84.76 %
N. unit vectors	46.94°	27.04°	44.54 %	81.33 %
Classification	35.07°	10.46°	72.26 %	84.86 %

Table 7.1: The performance of each method when tested on the validation dataset of 2000 images.

7.1 Heading Estimation

7.1.1 Prediction Accuracy on a Class by Class Basis

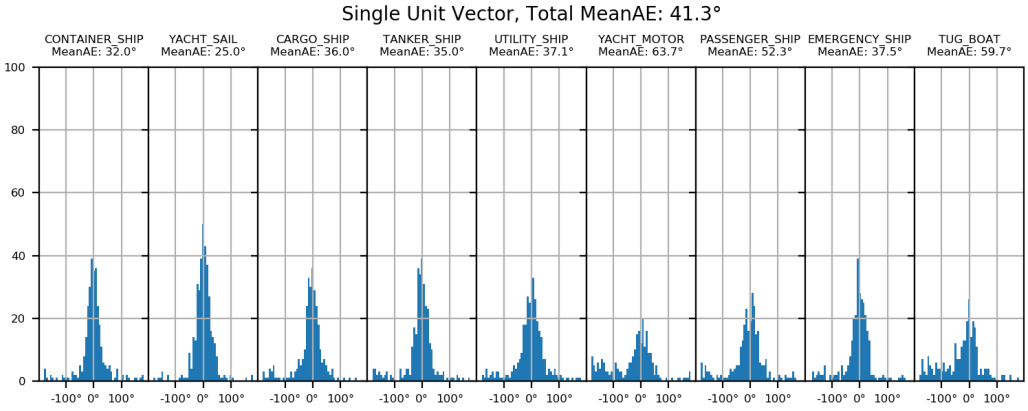
From Table 7.1 it is clear that the classification-based method performed best on the validation dataset, followed by the single unit vector method, and then the N. unit vectors method. It is interesting that according to the Acc90 metric the single unit vector approach performs almost as well as the classification-based approach. This implies that the classification-based approach does not reduce the number of big errors, but rather makes the 'normal' predictions much more accurate. This hypothesis is backed up by the Acc22.5 and medianAE metrics, in which it the classification-based method is clearly the best. See Section 3.3 for information about these metrics.

Somewhat surprisingly, the single unit vector approach performs better than the N. unit vector approach. During development it was theorized that this was due to the N. unit vector approach model not being trained for long enough. This is because, on a per training step basis the N. unit vector approach updates its parameters less frequently than the single unit vector approach, since only the parameters corresponding to the detected class are updated[1]. To test this theory, a model employing this approach was trained for 2000K training steps, twice as long as the other models. However, this model did not show a notable improvement in performance. The current theory is that this lack of performance stems from its lesser ability to draw from generalized features in its predictions, but this theory needs further experimentation to be proven or disproved.

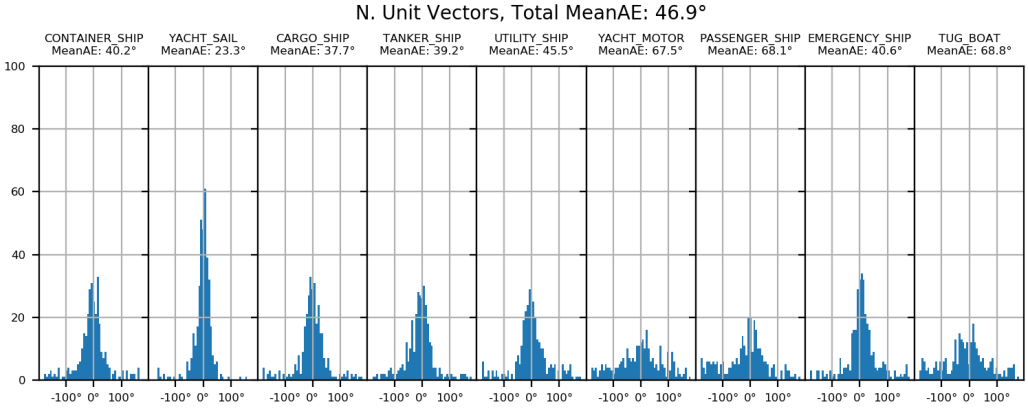
As noted earlier in the thesis, the Mask RCNN models trained in these experiments are trained to detect several different classes. It is therefore of interest to examine how the heading prediction accuracy varies for the defined classes. Figure 7.1 illustrates this through a series of histograms showing the distributions of heading estimation errors around the ground truth headings per class. The figure also shows the total meanAE and on a class by class basis.

By examining the histograms several things become clear. Firstly, the classification-based reformulation performs much better than the two other proposed methods. This is seen in the histograms as a higher concentration of samples close to the centers, indicating predictions with low angular errors. As noted previously in this section, the classification-based method in general has much more precise predictions, but the same amount of large errors. This is also visible in the histograms, in which the distribution peaks produced by the classification method are thin and tall. From the histograms it is also clear that some classes are more difficult to predict than others. The tug boat and motor yacht classes seem to be problematic for the unit vector based methods, while the classification-based method still performs well, even on these difficult classes. It is difficult to

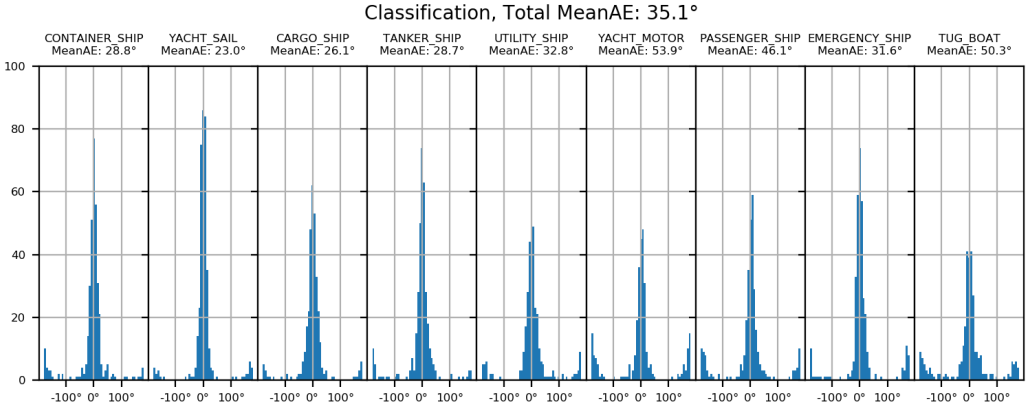
[1] In the final vector prediction layer.



(a)



(b)



(c)

Figure 7.1: Histograms illustrating the distributions of heading estimation errors around the ground truth heading per class.

pinpoint why these classes are especially difficult, other than that they are both often small and that some of the tug boat models lack a clear directionality. The motor yacht class has proven to be difficult to detect in some images, but this should not influence the result too much, since the models have to detect the object before they can predict a heading. This naturally restricts the most difficult instances from being counted in these histograms, since they would not be detected by the model in the first place. The histograms also show that the sailboat class has the lowest corresponding absolute error, meaning that it is the easiest class with regards to heading estimation. This might be caused by a bias in the dataset, since almost all of the sailboat 3D models used in the dataset have the same sail geometry. In a future experiment it would be easy to include more 3D models of sailboats and reduce this effect. The feature attribution experiment in Section 7.2.2 examines this bias through feature attribution.

The classification-based method, shown in Figure 7.1c produces characteristic angular error distributions in the histograms. Whereas the other two models mainly create distributions with a single peak at around zero error with respect to the ground truth, the classification-based method often creates distributions with two peaks. One at zero error, and another one at $180°$ error. Actually, in the figure it often creates three peaks, but due to the symmetry around $180°$ and $-180°$ these two peaks are actually a single peak split into two. A clear example of this behaviour can be seen in the histogram corresponding to the motor yacht boat class. This effect is likely due too the nature of the classification formulation and will be further examined in Section 7.1.6.

In [25], which introduced the heading estimation methods used in this thesis, the researchers tested their models on two datasets. One of which being the EPFL: Multi-View Car Dataset[76], which contains images of cars taken at an automotive show, with 2358 training samples and 1120 testing samples. Similarly to the results achieved in this thesis, the classification-based method performed the best, achieving MeanAE score of $9.86°$ and a MedianAE score of $3.14°$. Even though these models were tested on a different dataset than the Mask RCNN models developed in this thesis, it is of some interest to compare their achieved performance with the performances of the methods implemented in this work. There are likely several causes for the difference in achieved performance. To start, their models, which similarly to the Mask RCNN models in this thesis uses ResNet-101 as a feature extractor, only perform heading estimation. In contrast, the Mask RCNN networks in this thesis are much more busy, performing object detection, classification, bounding box regression, mask regression, and heading estimation. The EPFL dataset appears to be a simpler dataset as well, as all the images have their subject clearly visible in the middle of the frame and are taken from similar viewpoints. Some sample images from the EPFL dataset are shown in Figure 7.2. Additionally, as noted in Section 3.2.3, the performance of the heading estimation depends on which feature maps the heading is estimated from. The heading estimation modules implemented in this thesis are connected to the final feature map of the feature extraction network, which is probably not ideal as discussed in Section 3.2.3.

Figure 7.2: Some sample images from the EPFL: Multi-View Car Dataset. The figure is from [25], with some modifications.

7.1.2 Apparent Heading vs. True Heading

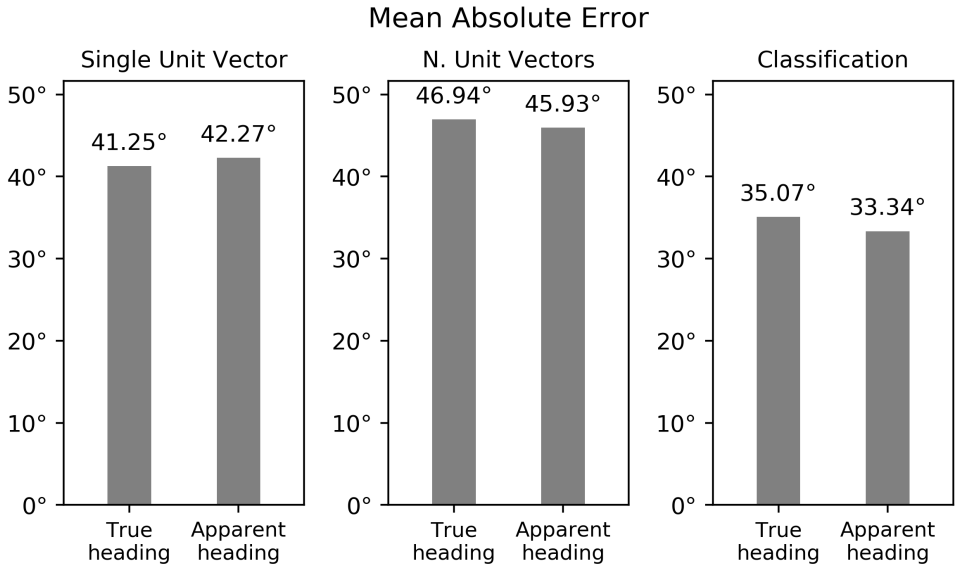
As illustrated in Section 5.6, the apparent heading of an object in an image changes as the object moves horizontally in the frame. To test whether it is better to estimate the apparent heading instead of the true heading, each of the models had an identical twin model which was trained to estimate the apparent heading instead of the true heading. Then, all six models were tested on the validation dataset consisting of 2000 images. Figure 7.3a and Figure 7.3b shows the resulting performance metrics. Note that the metrics generated for the models estimating the apparent heading used the apparent heading as ground truth. This means that the models did not have to transform their heading estimates to true heading via Equation 5.4 which would likely further increase their errors, since errors in bounding box estimation would now influence the heading predictions.

Figure 7.3a and 7.3b implies that estimating the apparent heading instead of the true heading does little or nothing at all to improve the accuracy of the predictions, especially if one takes into account the likely induced errors from transforming the heading predictions back to true heading, via Equation 5.4. The lack of improvement might be because the models are not accurate enough for the improvement to be distinguishable from noise. This theory is supported by the fact that the classification method, which has the clearly highest performance, shows the greatest improvement when estimating the apparent heading instead of the true heading. Alternatively, the networks might have learned to cope with the visual effect of apparent heading through some alternative mechanism, or maybe the camera field of view in the synthetic datasets are too narrow for this effect to have a degrading effect on the predictions. The camera field of view in the synthetic datasets is restrained to between 30° and 80°. In a future experiment another dataset with a wider field of view could be made to further examine this.

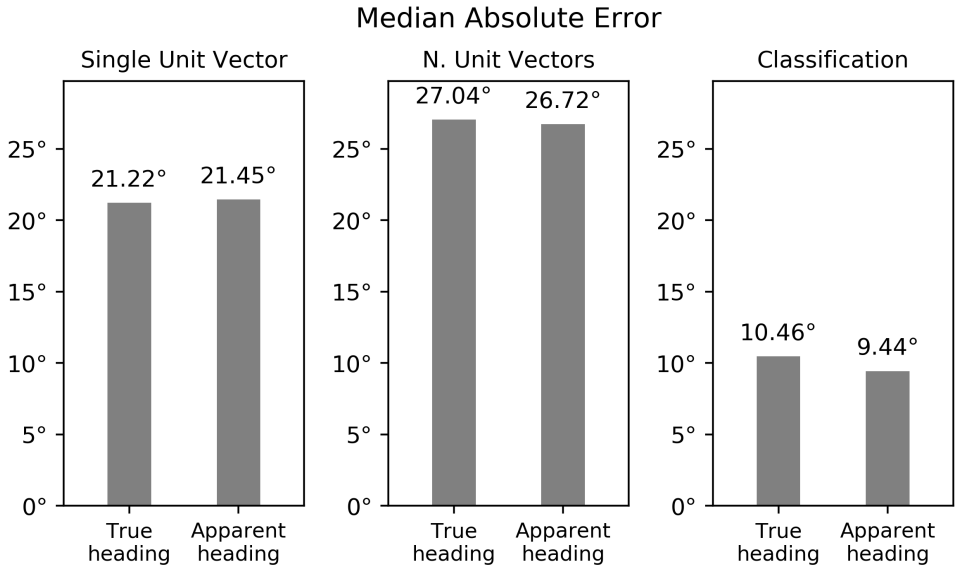
Another way to verify whether estimating the true heading induces errors in the predictions is to examine the prediction's absolute angular error as a function of the object's horizontal position in the image frame. If the effect reduces the model's precision, the models that estimate the true heading directly should have larger angular errors as the objects move towards the edges of the image, while the models estimating the apparent headings should have constant errors. Keep in mind that objects toward the edges of the images, with large horizontal offsets, might not be entirely contained within the image. This could also lead to an increase in the absolute errors. Figure 7.4 illustrates this relationship for both of the classification-based models. It is difficult to see a clear discrepancy between the two plots. A potential pattern might become more clear if the validation dataset included objects with higher horizontal offset, which would make the visual effect stronger. However, as shown in Figure 7.4, the plots do not indicate that the models estimating the true heading suffer much from a degrading effect caused by the apparent heading of the vessels.

7.1.3 Prediction Accuracy as a Function of Object Distance

Figure 7.5 highlights the angular errors as a function of the object distance, split by class. For clarity, only the predictions produced by the classification-based method are shown. The plots produced by the other two methods are very similar, but with more vertical spread due to their lower accuracy.

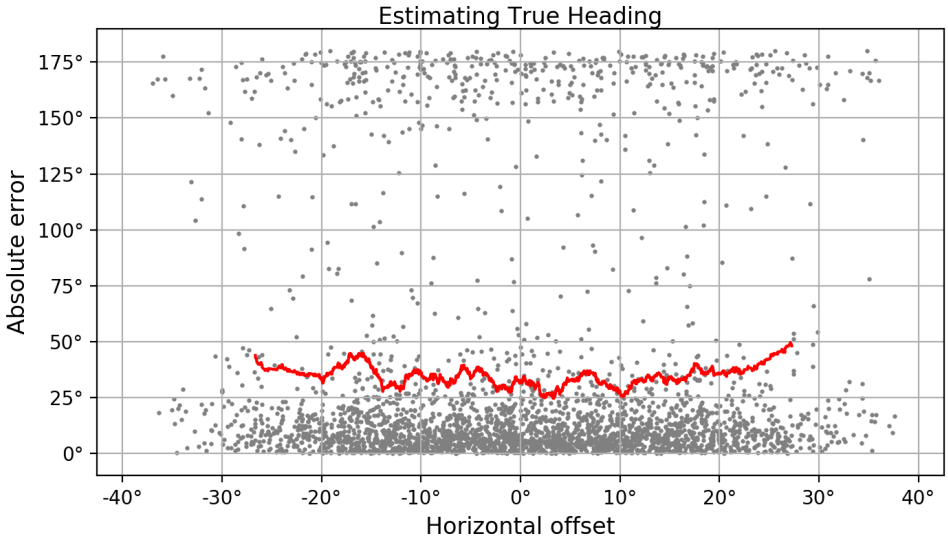


(a)

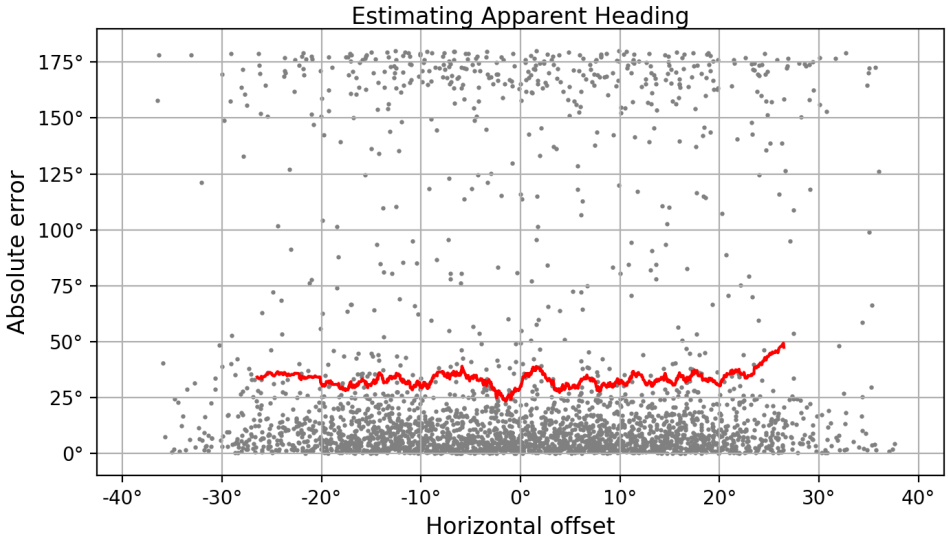


(b)

Figure 7.3: The meanAE and medianAE scores for all three methods when using true heading and apparent heading. The models were tested on the entire validation dataset, which consists of 2000 images



(a)



(b)

Figure 7.4: The absolute errors as a function of horizontal object offset from the center of the image frame. The heading predictions have been generated by the classification-based method The red line has been smoothed by a convolutional box filter of size 200.

Immediately it is clear that the vessel distance distributions are not equal for each class. Classes such as motor yacht or emergency ship are almost never farther away from the camera than about 400 meters. This is due to the way the synthetic dataset generation software handles small boats. Quickly summarized, it forces small boats to be closer to the camera, where it is more realistic to expect the Mask RCNN model to be able to detect them. Section 5.5 goes into further detail on this topic. It is also clear that the maximum distance allowed from the camera is 1000 meters. This is due to how the synthetic dataset generation was configured.

Back to the topic of the prediction accuracy. It seems the prediction accuracy is stable with respect to object distance. This is good, since it indicates that the model has learned to base its heading predictions on features that are robust to scaling. The plots make it is clear that the classification-based method often produces prediction-error distributions with two peaks, one around 0° angular error and the other at 180° angular error. This will be discussed further in Section 7.1.6.

7.1.4 Prediction Accuracy as a Function of Object Heading

Figure 7.6 shows how the prediction accuracy of the three methods varies with the ground truth heading of the detected object. Remember that a heading of 0° means that the vessel points directly away from the camera, while a heading of 180° means that the vessel points directly towards the camera. The graphs describing the absolute errors are smoothed versions of the raw prediction errors, similarly to how the graph in Figure 7.5 was created. From the figure one can observe that for most classes and ground truth headings the classification-based method is, as expected, the most accurate. As noted in previous sections, this result might also indicate that the models suffer from a bias stemming from the sailboat's sail geometry, since they all perform exceptionally well on this class in particular.

It is clear that the models often struggle to produce accurate heading predictions when the detected vessel points in the parallel or opposite direction of the camera. Intuitively this makes sense since many distinct features of the vessel are not visible in this configuration, such as the vessel's bow or stern. If the models base their predictions on these features, it is natural that their prediction accuracy will suffer when they are not visible. When a vessel points directly towards or away from the camera, its geometry is often also radically different than it is when seen from its side. Clearly this might also degrade the performance. This effect can be seen in the tanker class, where all three models produce less accurate predictions when the ships point directly at or away from the camera. The models produce predictions of the highest quality when the detected vessels are orthogonal to the camera. This is also intuitive, since this orientation causes most of the vessel's distinct features to be visible.

Otherwise the models seem to be quite robust with respect to the ground truth heading - at least if one ignores the single unit vector method's struggle with the motor yacht class. This is a good result since it indicates that the models have learned to base their heading prediction on features that are somewhat robust to changes in the ground truth heading. If these models were to be used in a real world application, they could be expected to perform reasonably well regardless of the ground truth heading of the detected vessels. Of course the models would have to undergo extensive domain

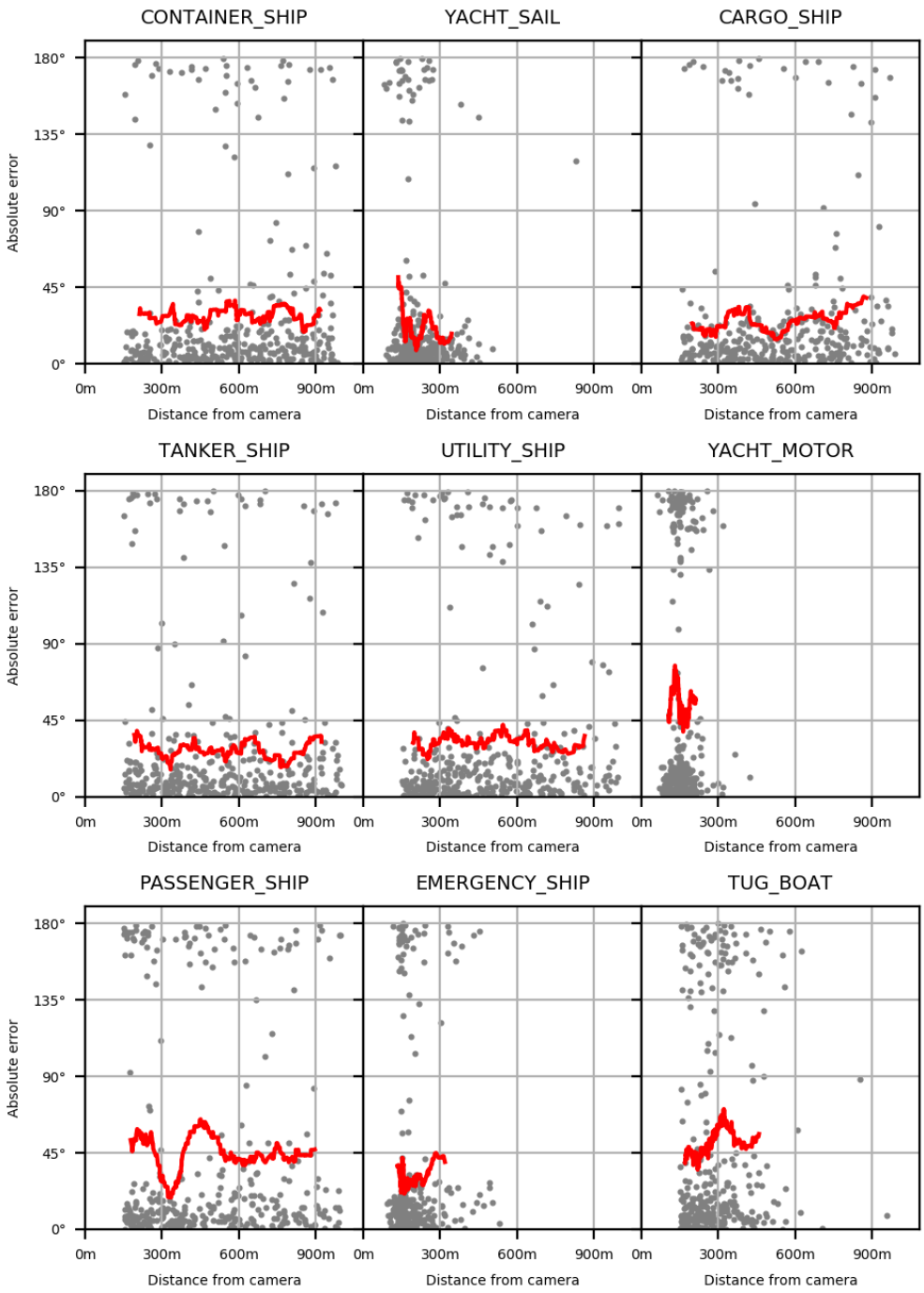


Figure 7.5: The angular prediction error as a function of the detected object's distance. The classification-based method generated the predictions shown in this graph. The red line has been smoothed by a convolutional box filter of size 50.

adaptation or be trained on a real world dataset first.

7.1.5 Comparison with Unmodified Mask RCNN Architecture

As we modify the Mask RCNN network to perform more tasks than the initial classification, bounding box prediction, and mask estimation, the network's performance is expected to degrade. Figure 7.7 highlights how the mAP scores produced by the various models, each with a different heading prediction modification, compares to an unmodified Mask RCNN network trained and tested on the exact same dataset.

From Figure 7.7 it is clear that the performance of the object detection does indeed decrease, but not by as much as one might expect, as it still remains reasonably high for all three modified models. This might be because the models learn to base their classification predictions and heading predictions on the same features in the feature maps, leading to only a slight degradation of the initial classification prediction performance. In that case, this would be a nice example of multi-task learning. As noted in Section 6.2, the degradation of classification performance could probably be controlled by multiplying the loss function describing the heading loss with a constant.

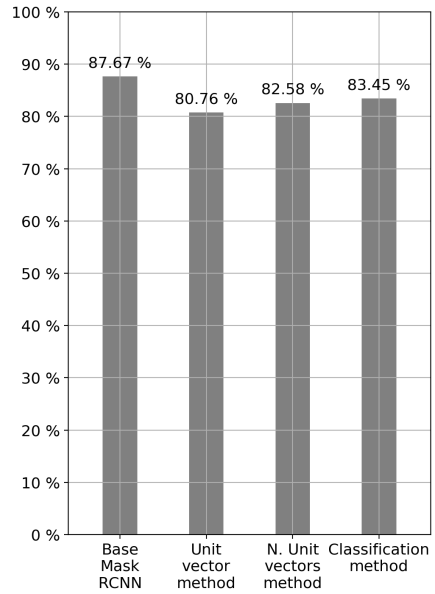


Figure 7.7: The mAP scores of the various Mask RCNN methods when tested on the validation dataset consisting of 2000 samples. The scores are calculated by the interpolated mAP algorithm discussed in Section 2.3.13 using a IOU threshold of 0.5.

It is also interesting that the method with the highest heading prediction performance has the least degraded object detection performance, if only by a small amount. This indicates that the increased heading prediction performance does not come at the cost of the rest of the system's performance. This could imply that the classification-based method is a more efficient use of network resources, at least with respect to the object detection performance, or that it is more compatible with the rest of the networks tasks, in that it bases its predictions on more similar features.

7.1.6 Method Specific Evaluations

As alluded to in Section 6.2.6, two method specific aspects will be presented and discussed in this section. First one out is the internal classification scores in the classification-based method. Three examples are shown in Figure 7.8, each highlighting a different aspect of the method. The scores are color-coded based on which of the classifications produced them. The ground truth is shown as

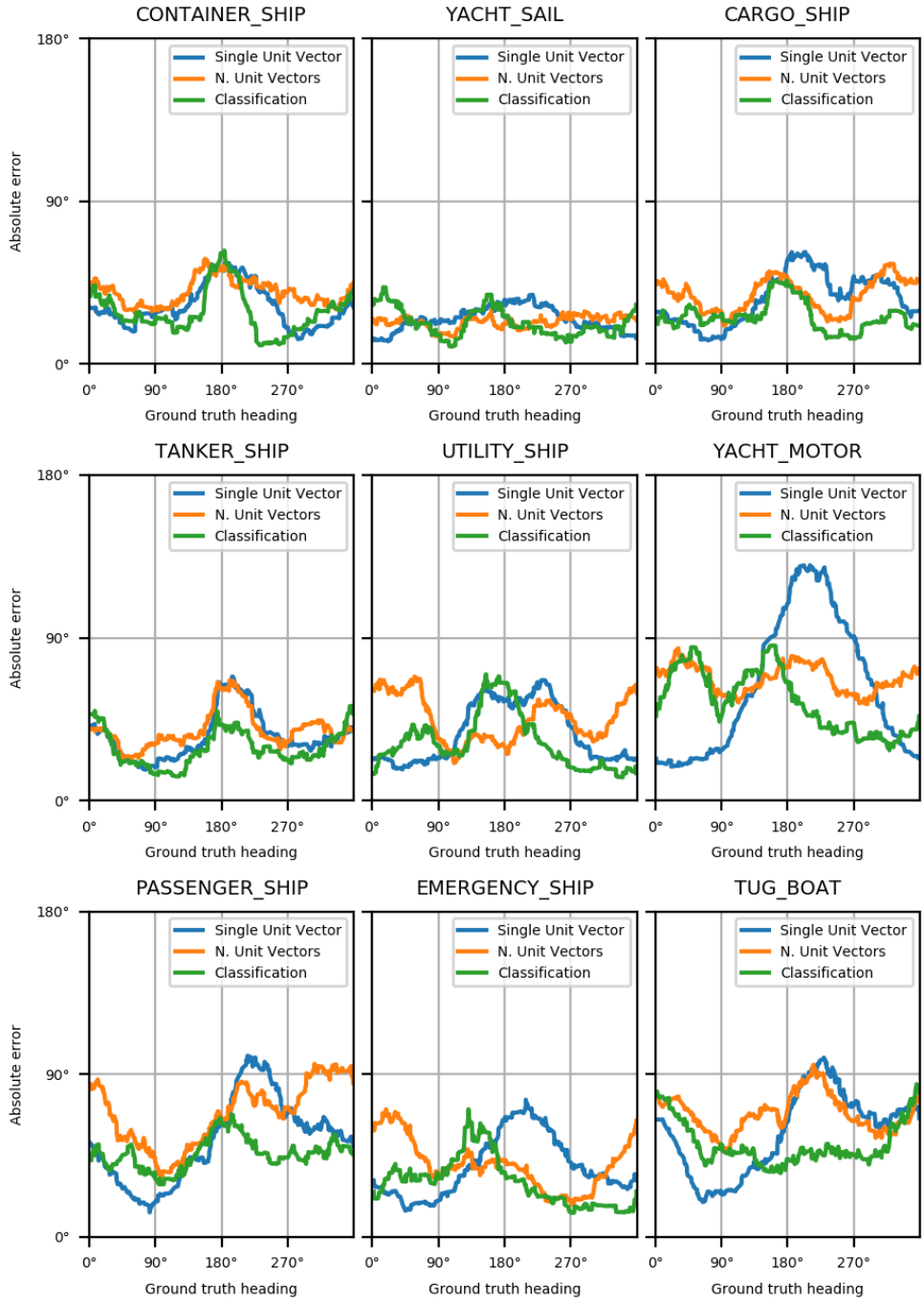


Figure 7.6: The angular prediction error as a function of the detected object's ground truth heading. The graphs have been generated by smoothing the sample results with a convolutional box filter of size 50.

a dark line, and the final predicted heading is represented by a light grey line.

The first example, shown in Figure 7.8a and 7.8b, represents how the method is supposed to work, even though the predicted heading is a little different than the ground truth heading. The predictions are for the ferry in the middle of the image and it can be seen that there is clear agreement between the classifications that the vessel is pointing towards the lower left of the image. The mean shift technique then finds the center of the cluster and reports it as the final predicted heading. There are two possible points of failure in this technique. The classifications can fail to predict the right heading, or the mean shift technique can fail to find the correct heading through its clustering process. In this case, the clustering component works perfectly, and the small error is caused by the classification network's failure to accurately predict the vessel's heading.

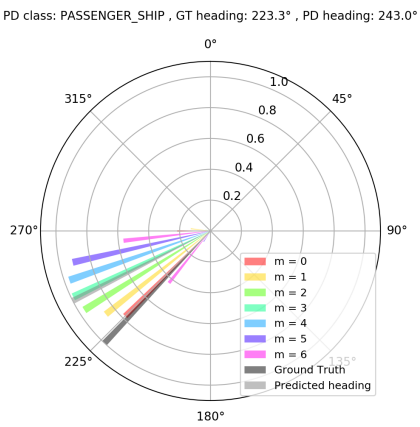
The second example, shown in Figure 7.8c and 7.8d, shows the method predicting the heading of a container ship. The distance to the vessel, as well as the fact that it is partly occluded by the supply ship, makes this a more difficult instance than the previous one. It is clear from the internal classification scores that the model is not quite sure which way the vessel in pointing, since the discrete headings have low classification scores and are quite spread out. However, it is still able to produce a quite accurate prediction after the mean shift clustering technique is applied. This example shows the benefits of ensemble based methods. Even though the individual classification scores are not very accurate, the 'wisdom of the crowd' effect still leads to an accurate final prediction. Of the three methods of heading prediction tested in these experiments, this effect is unique to the classification-based method.

The third and final example, shown in Figure 7.8e and 7.8f, shows a typical erroneous prediction. As is visible in Figure 7.1c, this method's errors are often 180° from the ground truth heading. This is also the case in this example, as the method predicts the heading of the vessel to be in the complete opposite direction of the ground truth. The classification scores indicate that the model was not sure which way the vessel was pointing. There are two distinct clusters in the discrete heading classifications, one in the correct direction and one in the opposite direction. After inspecting several different predictions and the corresponding classification scores it has become clear that this pattern is often repeated and is probably responsible for most of the large erroneous predictions.

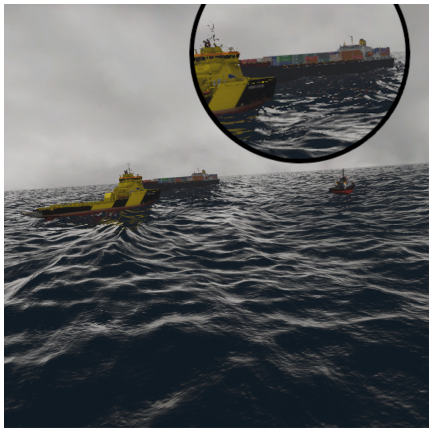
Since XAI and interpretability is a central topic in this thesis it is worth noting how these visualizations impact the model's interpretability. It is likely that an eventual end-user's trust in the models predictions would increase if he or she was presented with these scores alongside the predicted heading, as they offer insight into the models internal thought process. It is also much easier to detect when the classification-based model is uncertain in its predictions than it is for the two other methods discussed in this experiment. This trait, even though difficult to measure empirically, is very positive.

The second method specific aspect that this section will examine is the distribution of heading vectors produced by the N. Unit vector method. As a quick reminder, this method predicts heading vectors for all defined classes and then samples the one corresponding to the predicted class as the final predicted heading. Figure 7.9 shows two example images, along with their predicted heading vector distributions.

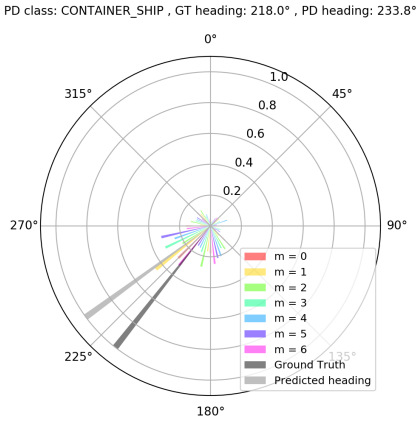
(a)



(b)

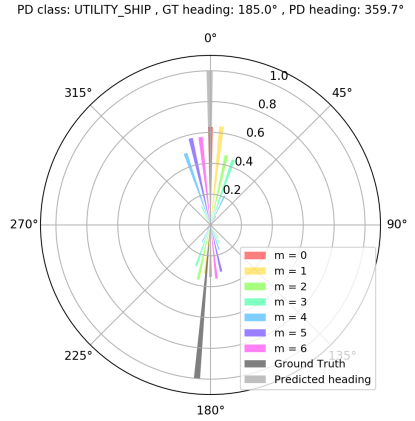


(c)



(d)

(e)



(f)

Figure 7.8: The classification-based heading estimation method's internal classification score distributions, and the images that produced them.

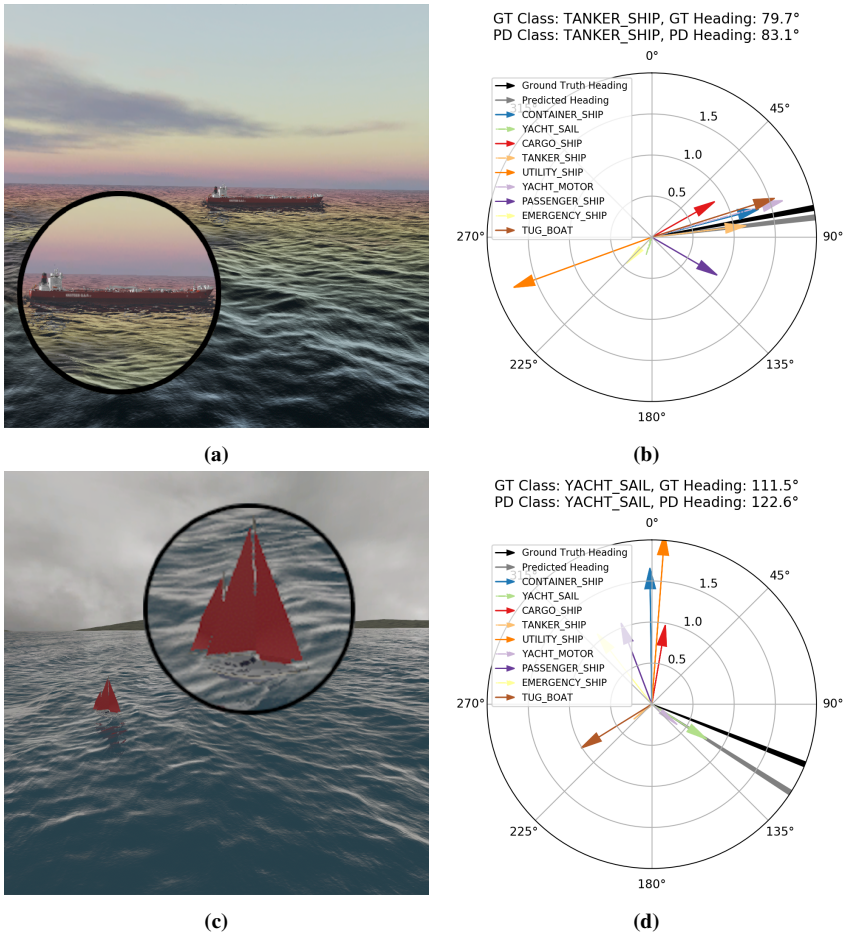


Figure 7.9: Some distributions of heading vectors produced by the N. Unit vector based heading estimation approach and the images that they are based on.

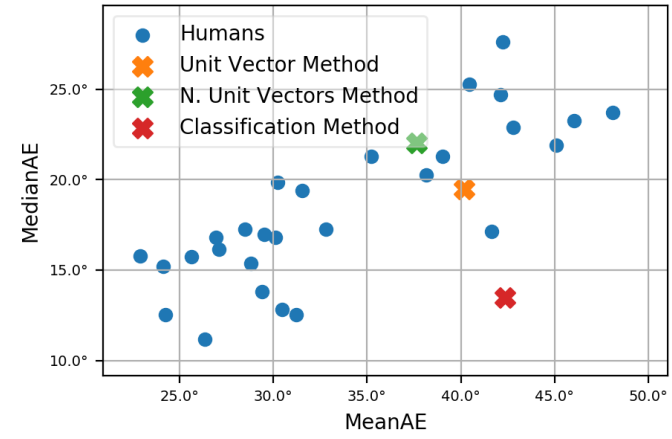
The first example, shown in Figure 7.9a and 7.9b, shows the distributions of heading vectors generated by the model while examining a tanker. The figure shows that many of the vectors point in the same direction. The vectors for the classes container ship, tug boat, motor yacht, cargo ship, and passenger ship all seem to point in a similar direction as the one for the tanker, which was selected as the final prediction since the model predicted the class tanker. While the heading vector generated for the class utility ship points in the opposite direction. This is interesting since utility ships, in the real world often called supply ships, have their superstructures in the opposite end of the ship compared to tankers. If the model used the superstructure in its predictions it is then natural that the heading vector for this class would point in the opposite direction, as the superstructure's position in relation to the vessel's hull would imply that it was pointing in the opposite direction. This indicates that the model is able to use class specific characteristics in its heading predictions. This is likely made possible by the network's ability to predict class specific heading vectors.

The second example was chosen to highlight a potential issue with this approach. Since the model samples from the set of predicted heading vectors based on the predicted class, an error from the classification network will propagate to the heading estimation system. Observe how, in Figure 7.9c and 7.9d, many of the predicted heading vectors point in radically different directions than the one representing the predicted class. If the model had predicted the wrong class, the error would cause the heading estimation system to select one of the other heading vectors, inducing a large error. This effect could be especially destructive if the predicted heading vectors were based on very different, class specific features. The class specific headings would then likely point in very different directions and an error in the classification could as a result induce a large heading estimation error.

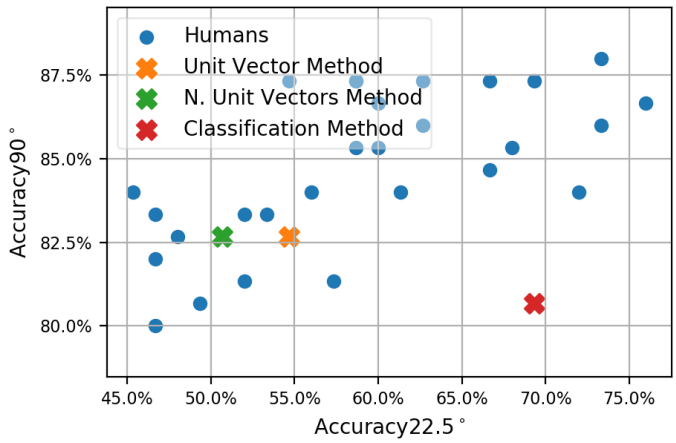
7.1.7 Comparison to Human Performance

As discussed in Section 6.2.7, a survey of humans' abilities to estimate the heading of maritime vessels was conducted. This section will conclude the experiments on heading estimation by going through the results of that survey and comparing the performance of the models implemented in this thesis to humans. A limiting factor in this experiment was that out of the 90 samples included in the survey, only 75 of them were detected by all three Mask RCNN models and could be used in the evaluation. Most of the missed samples belonged to the 'hard' sample category. In the models' defence, many of the participants would likely have struggled to spot some of these vessels as well, if not for the red bounding box marking their location, as they were either very small in the image frame or heavily occluded.

The results shown in Figure 7.10a and 7.10b indicate that the models perform slightly worse than the average human. After analysing the figures a few notable things have become apparent. Firstly, if we compare the model's performances with the ones in Table 7.1 it is clear that the classification-based method under-performed slightly, while the N. Unit Vector based method over-performed. It is unlikely that this is caused by anything else than noise due to the low sample number being analysed. The classification-based method achieved a very good medianAE score on par with the best humans, but was let down by its high frequency of large errors, indicated by its relatively low accuracy90 score. Its performance is unlike any human tested in that most of its predictions were either very accurate or entirely wrong. Most of the humans were either very accurate with few large errors, or inaccurate with many large errors.



(a)



(b)

Figure 7.10: The three methods' performances compared to how humans performed. Out of the 90 test samples, 75 was detected by all three models and was used to calculate these values.

During the analysis of the results a concern that some participants did not understand that they were meant to predict the true heading of the vessels arose. To check whether this was the case, the participants' scores if evaluated with respect to the vessels' apparent heading was computed. The resulting change in performance is shown in Figure 7.11. As expected the participants scores differed slightly when evaluated with respect to the apparent heading instead of the true heading. Many of the best results got decidedly worse, indicating that they had understood their task correctly. Some participants' scores increased, while others' barely changed. In the end, it was decided that most, if not all, participants had correctly understood their task.

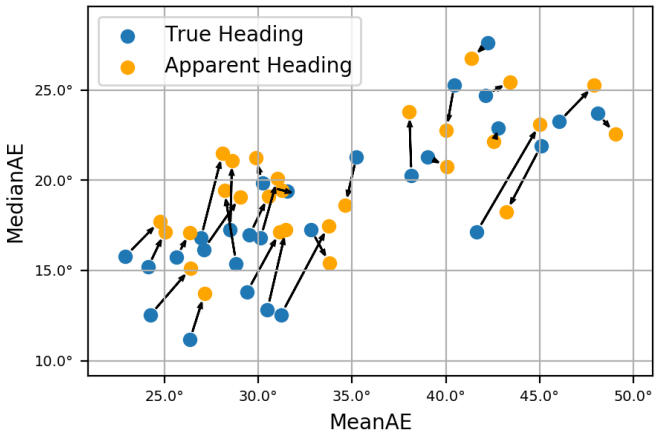


Figure 7.11: A concern was that some participants in the study did not understand that they were meant to estimate the true heading of the vessels. This figure illustrates how the participants' scores would have changed if their predictions were measured with respect to the apparent heading instead of the true heading.

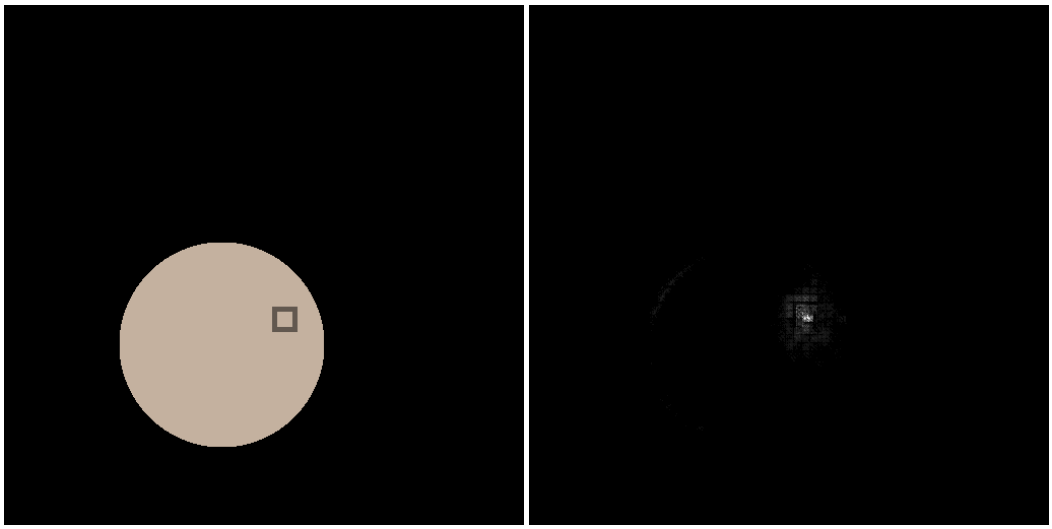
7.2 Feature Attribution on the Heading Estimations

This section will discuss the experiments described in Section 6.3.1 and 6.3.2 which modifies the Integrated Gradients method and the LIME method to generate explanations for heading estimations. Before they are applied on models trained on the maritime dataset in Section 7.2.2 they are validated and tuned on models trained on a simpler dataset of circular blobs with a single feature defining their heading.

7.2.1 Validating the XAI Methods

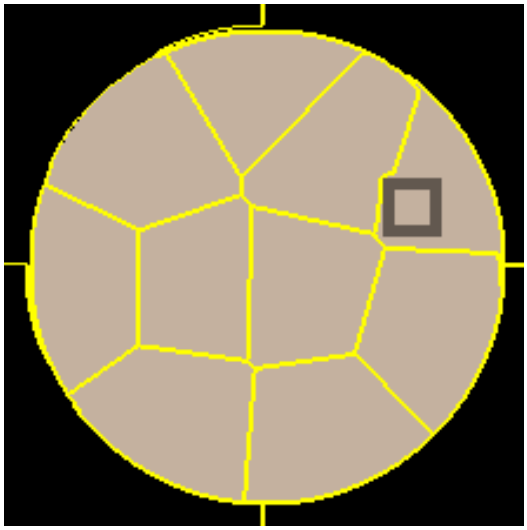
To illustrate the validation process and resulting performance of the two XAI methods a few sample images will be presented, as well as the explanations generated by the two methods. Two potential merging functions for the Integrated Gradients based method will be compared. Merging functions are discussed in Section 4.5.3. To illustrate the LIME based method's performance, the fully segmented image as well as the top three most influential segments will be presented. Keep in mind that this modified LIME version crops into the detected object's bounding box to achieve higher fidelity attribution. The details surrounding how the LIME method has been modified for this experiment is discussed in Section 4.5.5.

The training process and final performance of the Mask RCNN model trained on the 'Blob World' dataset is not very important for the discussion in this section. It was however, able to accurately predict the heading of the blobs. This implies, as discussed in Section 6.3.1, that the Mask RCNN model has learned to use the dark surface-feature on the blob to estimate its heading, since the blob's heading is solely defined by the surface-feature's position.

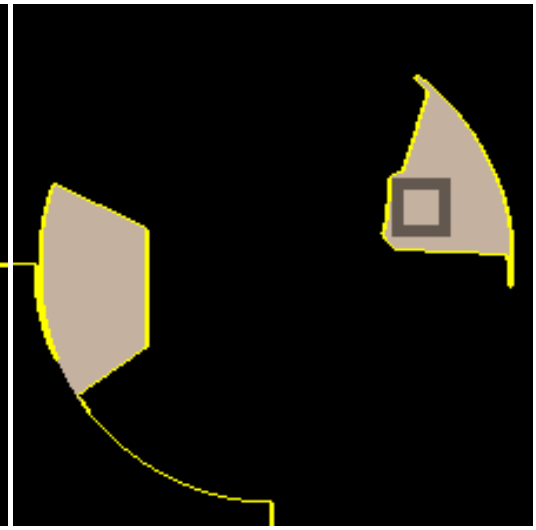


(a) The beige blob's heading is to be explained.

(b) The pixels highlighted by Integrated Gradients.



(c) The segmented blob available to LIME.



(d) The top three segments highlighted by LIME.

Figure 7.12: Feature attributions generated for a prediction on the 'Blob World' dataset.

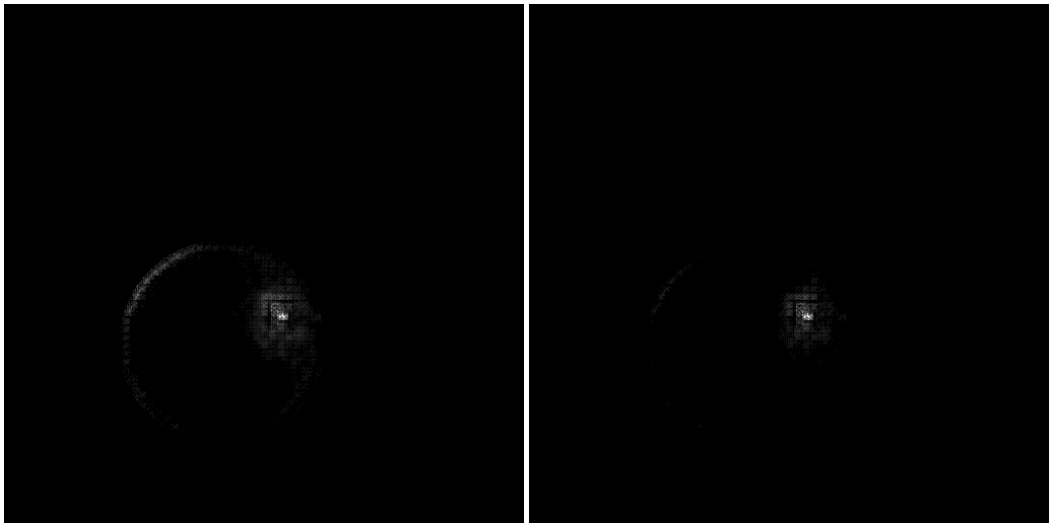
Figure 7.12 shows a typical feature attribution result. The target for the explanations is the heading prediction for the beige blob in Figure 7.12a. Figure 7.12b shows the pixels that the modified Integrated Gradients method highlights as being the most influential. The brighter the pixels are, the more influential the Integrated Gradients method deems them. The parallel component merging function was used to generate these attributions. It is expressed in Equation 4.5. In general it achieved the highest precision attribution of the two methods tested, this will be discussed in depth shortly. From the figure it is clear that the Integrated Gradients based method is able to highlight the heading defining feature, as the blob's feature is highlighted.

The two lower sub-figures in Figure 7.12 show the segmented image and the top three most influential segments as deemed by the LIME based method. From the figure it is clear that the image available to the LIME based method has been cropped to more closely fit the blob. This means that it only generates attributions in the close proximity of the detected object. This simplifies the training of the surrogate model, but obviously limits which parts of the image the LIME method can highlight. This trade-off is further discussed in Section 4.5.5. As is clear from Figure 7.12d, the LIME based method correctly highlights the desired feature in one of its three most influential segments.

After examining a set of feature attributions generated by both methods, some general observations can be shared.

The Integrated Gradients based method performs better than the LIME based method in nearly all cases. Its better performance can be highlighted through two main advantages. To start, its feature attribution is more accurate. It almost always highlights the correct feature, while the LIME based method is sometimes unable to do so. The second advantage is how much less tuning the Integrated Gradients based method requires. It simply worked immediately, while LIME required significant tuning before it was able to produce accurate feature attributions. This is unfortunate because of two reasons. To start, it is very time consuming to perform this tuning. During the development of the systems in this thesis, several days were spent tuning the LIME based method. The second reason why requiring extensive tuning is bad is because it is not known whether the method will be able to generalize to other datasets without more tuning. This might mean that the time spent tuning the method on the 'Blob World' will do nothing to help it perform on the maritime dataset. The Integrated Gradients based method is more likely to generalize well, since it worked without any tuning. Unless the initial version of the Integrated Gradients based method was perfectly tuned for the 'Blob World' by chance. Fortunately, this is unlikely. One potential reason for the LIME based method requiring a lot of tuning before it was able to adequately perform might be that it is now explaining heading predictions instead of classification predictions. The method could in that case be expected to generalize well to heading predictions on other datasets, since it has now been adapted to explain heading regressions. On the other hand, the method's configuration could now be overfitted on the blob dataset, which would lead to degraded performance on other datasets.

As discussed earlier in Section 4.5.3, the Integrated Gradients based method can be configured with several different merging functions. As a reminder the task of the merging function is to combine the two-dimensional gradient of the heading vector derivatives with respect to the input pixels into a single value that can be highlighted in an image-pixel. Figure 7.13 compares the attributions created by the L2-norm based merging function with the attributions from the parallel decomposition



(a) Attributions by the L2-norm merging function.

(b) Attributions by the parallel decomposition merging function.

Figure 7.13: The attributions generated by two different merging functions. Both explain the predicted heading of the blob in Figure 7.12a.

based function. Both of these attributions explain the heading prediction of the blob in Figure 7.12a. These merging functions are depicted in Equation 4.4 and 4.5 respectively. From the figures it can be seen that the attributions created by the parallel decomposition based function is less noisy than the one created by the L2-norm based function. This is likely due to how the parallel decomposition based function filters out gradients that do not contribute in the direction of the actually predicted heading, while the simpler L2-norm based method treats all gradients equally. When examining the attributions generated for several predictions, the attributions from the parallel decomposition merging function was found to be less noisy in most cases. The method used 50 steps to approximate the curve interval from the baseline input to the actual input. The baseline input was set to be a black image.

As mentioned earlier, the LIME based method required extensive tuning before it is able to achieve acceptable feature attribution. The details around this final tuning configuration will now be discussed. Figure 7.14 shows an explanation generated by the LIME based method when configured to provide higher resolution attributions by having more segments to choose from. As previously mentioned, this makes the attribution problem more difficult, but offers higher fidelity attributions if the system performs well. In the end, the best performing LIME configuration used 1000 perturbed samples to train its surrogate model. The defined similarity kernel function, which prioritizes the perturbed samples based on a distance measure from the original unperturbed sample, was defined as

$$\pi(\mathbf{z}, \mathbf{x}) = \sqrt{\exp \frac{-D(\mathbf{z}, \mathbf{x})^2}{\sigma^2}} \tag{7.1}$$

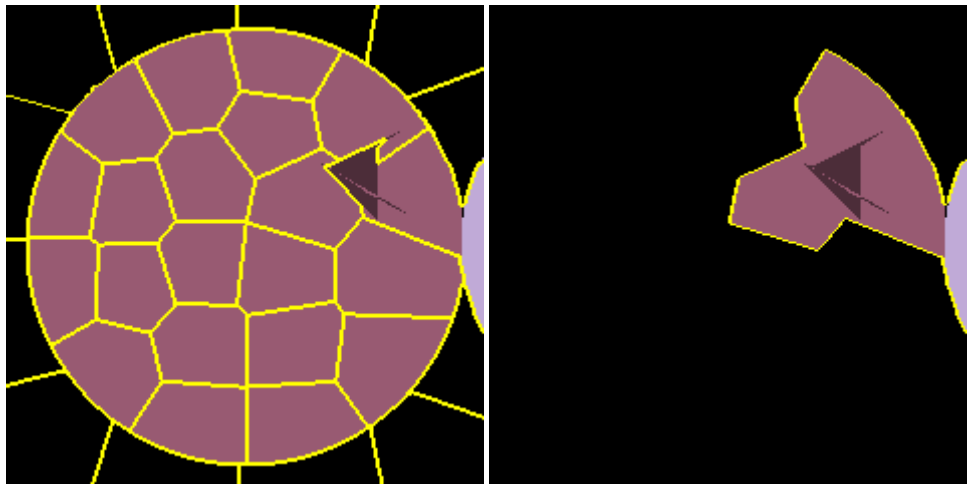
with $\sigma = 0.1$ and $D(\mathbf{z}, \mathbf{x})$ being a distance measure between the vector of perturbed image segments $\mathbf{z}$ and the vector of original segments $\mathbf{x}$ defined as

$$D(\mathbf{z}, \mathbf{x}) = \frac{\sum_{i=0}^{n} z_i x_i}{\sqrt{\sum_{i=0}^{n} z_i^2} \sqrt{\sum_{i=0}^{n} x_i^2}} \tag{7.2}$$

also called the cosine distance between vector $\mathbf{z}$ and vector $\mathbf{x}$. The final mapping function used was

$$m(\mathbf{h}, \mathbf{h}_{new}, \mathbf{h}_{err}) = \frac{\frac{\mathbf{h}}{\|\mathbf{h}\|} \cdot \frac{\mathbf{h}_{new}}{\|\mathbf{h}_{new}\|}}{1 + \left(\alpha \left\| \frac{\mathbf{h} \cdot \mathbf{h}_{err}}{\|\mathbf{h}\|} \frac{\mathbf{h}}{\|\mathbf{h}\|} \right\| + \beta \left\| \mathbf{h}_{err} - \frac{\mathbf{h} \cdot \mathbf{h}_{err}}{\|\mathbf{h}\|} \frac{\mathbf{h}}{\|\mathbf{h}\|} \right\|\right)} \tag{7.3}$$

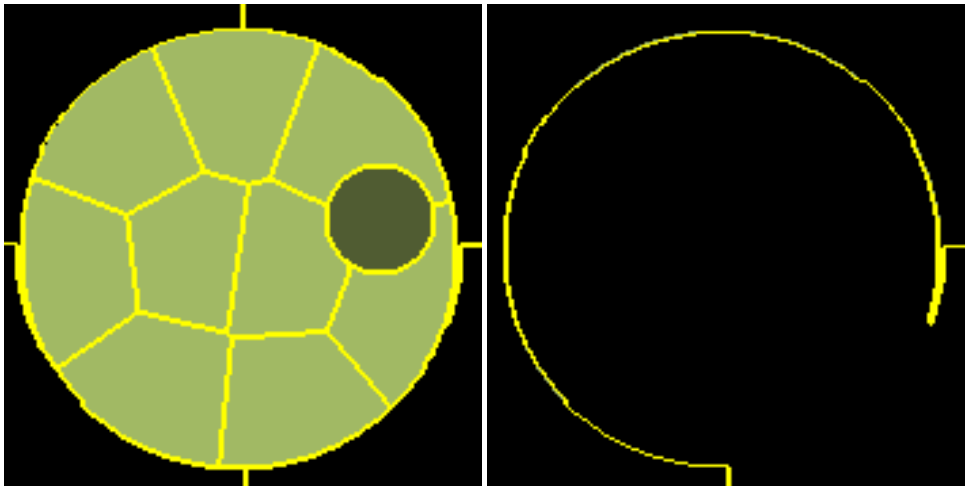
with $\alpha = \beta = \sqrt{2}$. Section 4.5.5 discusses potential mapping functions in depth. In short, their purpose is to convert a new heading prediction to an emulated classification score by comparing it to the original heading prediction. The image segments that were hidden in the perturbed samples were replaced by uniformly colored segment with the average color of the removed segment. The Simple Linear Iterative Clustering (SLIC) algorithm was used to segment the images, splitting the objects into 10-30 segments provided the best results. The method was then configured to highlight the three most positively influential segments. The final LIME configuration does not always provide accurate attributions, sometimes it simply fails to highlight the correct feature. Figure 7.15 shows a typical attribution error from LIME, where it fails to highlight the desired feature.



(a) The segmented blob available to LIME. **(b)** The top three segments highlighted by LIME.

Figure 7.14: LIME configured with higher resolution segments. This explanation used 2000 samples to enable the higher fidelity result.

As mentioned previously, the purpose of this experiment is to validate and tune the XAI methods before using them to gain insights into the Mask RCNN model trained on the synthetic maritime dataset. A conscious effort was therefore made to avoid fine tuning the methods too much to the



(a) The segmented blob available to LIME. **(b)** The top three segments highlighted by LIME.

Figure 7.15: A typical false result from the LIME based method. Observe that it fails to highlight the correct feature or even the blob, instead highlighting the background.

blob dataset, since it would be unlikely that the performance would generalize that well to the maritime dataset. To briefly summarize what was learned from this experiment, the Integrated Gradients based method performs better than the LIME base method in nearly all cases. It is very often able to highlight the correct features, while the LIME based method sometimes fails to do so. As a result, if later the integrate gradient based method and the LIME based method provide contradictory results, the LIME based method is likely incorrect.

In a future experiment this simple synthetic dataset could be expanded to more accurately analyse the performance of feature attributions systems. The blobs could be modified with false features, meaning features that are clearly visible but do not influence the defined heading of the blob. An accurate feature attribution system should then be able to ignore these and highlight only the features actually relevant to the blob's heading. On a system level, an analysing framework could be developed which automatically detects whether an attribution includes the correct feature and logs it. This would enable faster and more accurate evaluations of feature attributions systems than the current method based upon manual evaluation is capable of.

7.2.2 Applying the XAI Methods on the Synthetic Maritime Dataset

After having validated the performance of the two newly developed feature attribution methods on predictions that are defined by a single feature, it is time to see how they are able to perform on the much more complex ship dataset. As in the previous section, this section will not provide numerical analysis of the feature attribution performance of the systems. Doing so would require knowledge of which features the Mask RCNN model actually bases its predictions on, and unfortunately this information is not available. Determining these is the goal of this experiment. One thing that could

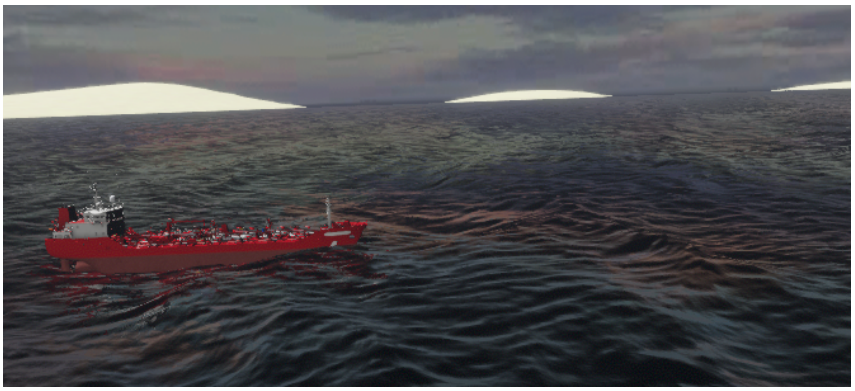
be expected if the feature attribution works correctly, is consistency between the feature attributions for similar predictions. In this case, similar predictions means predictions generated on images of ships of the same class and in similar orientations. Then, roughly the same features would be visible to the Mask RCNN model and it would be reasonable to expect a robust model to consistently base its predictions on the same features. Therefore the same features should be highlighted by the feature attribution methods between similar images. Another thing to look for is consistency between the two feature attribution methods' highlighted features. If they work correctly they should both be able to highlight the features the Mask RCNN model uses in its predictions, and thus they should highlight the same areas of the image.

When evaluating XAI methods, specifically methods for feature attribution, one has to be keenly aware of how confirmation bias might affect one's perception of a result. The results that are presented later in this section will be the ones that have been observed frequently enough that they are believed to be an actual feature of the feature attribution methods or Mask RCNN methods, not just a falsely perceived pattern in the noise.

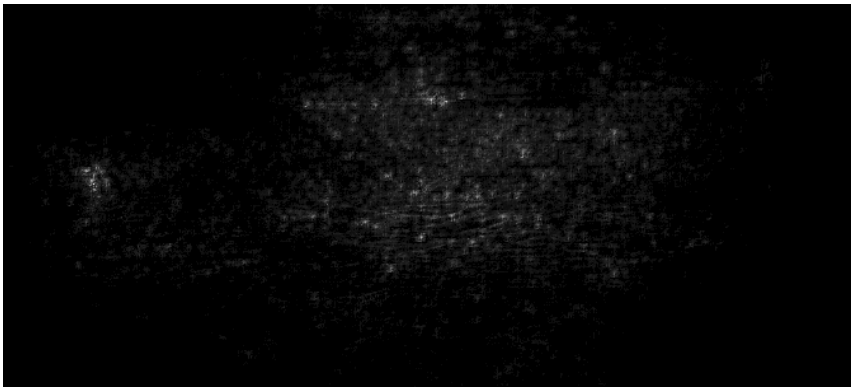
First, some general observations. Both methods struggled to consistently generate precise feature attributions on the synthetic maritime dataset. The methods often provided contradictory answers, failed to highlight specific features, or exhibited large amounts of noise. The Integrated Gradients based method often highlighted reasonable features of the vessel amongst the noise. It is possible that this result could be cleaned up by more extensive filtering. This is something that could be explored in further experiments. Figure 7.16 shows a typical feature attribution result created by the method based on Integrated Gradients. It is clear that the noise is drastically reduced when using the parallel decomposition merging function. In general, the parallel decomposition merging function often produced less noisy feature attributions. Observe that the cluster of highlighted pixels around the superstructure, and the less defined clustering around the ship's bow. It is also clear that the terrain in the background influences the feature attributions, either through the Mask RCNN model or the feature attribution method directly, since there are clusters of highlighted pixels in its area.

The LIME based method performed similarly on the synthetic maritime dataset as on the 'Blob World' dataset, only with more frequent false results. For comparison, Figure 7.17 shows which features of the vessel originally shown in Figure 7.16a the LIME based method highlights as most influential for the heading prediction. For this vessel the method actually produced a quite intuitive explanation, and highlighted the vessel's bow. Note that this is not always the case as the method often fails to highlight intuitive segments. Figure 7.18 highlights three such non-intuitive feature attribution results. Because of the frequency of unclear results from the LIME based method, the results from the Integrated Gradients based method will be used to analyse the behaviour of the Mask RCNN model. Although often quite noisy, its results are believed to be more accurate.

The Integrated Gradients based method frequently highlighted the superstructure of the detected vessel, if visible, in its feature attributions. This indicates the superstructure influences the Mask RCNN model's predictions. For humans this is an intuitive idea, since the feature is clearly defined and highly correlated with the actual heading of the vessel. The superstructure is likely to be a high-level feature internally in the Mask RCNN model feature maps, and since the heading prediction module is connected to the final feature map, it was expected to be able to utilize high-level



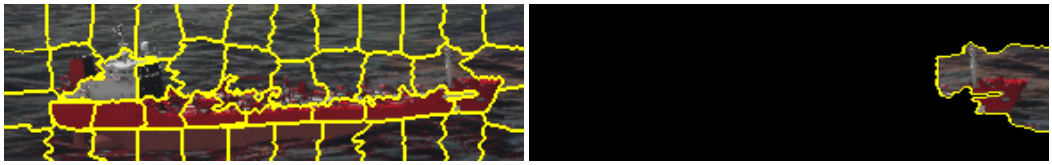
(a) The heading of the tanker was predicted to be $70°$, the ground truth heading was $87°$.



(b) The pixels highlighted by the Integrated Gradients based method with the l2-norm merging function.

(c) The pixels highlighted by the Integrated Gradients based method with the parallel decomposition merging function.

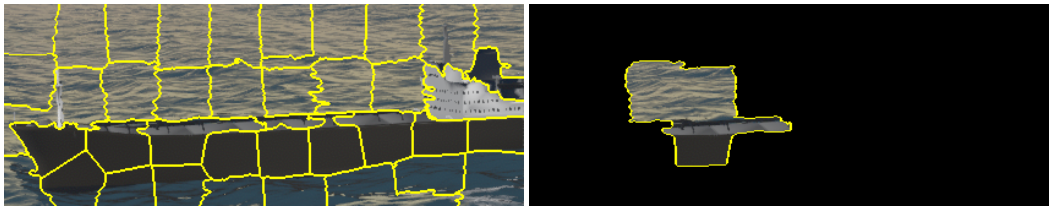
Figure 7.16: An example feature attribution by the Integrated Gradients based method. Note that the image has been cropped for increased visibility.



(a) The segments of the image available to the LIME based method.

(b) The top three segments highlighted by LIME.

Figure 7.17: The features highlighted by the LIME based feature attribution method.



(a) Detected vessel #1.

(b) Feature attribution #1.

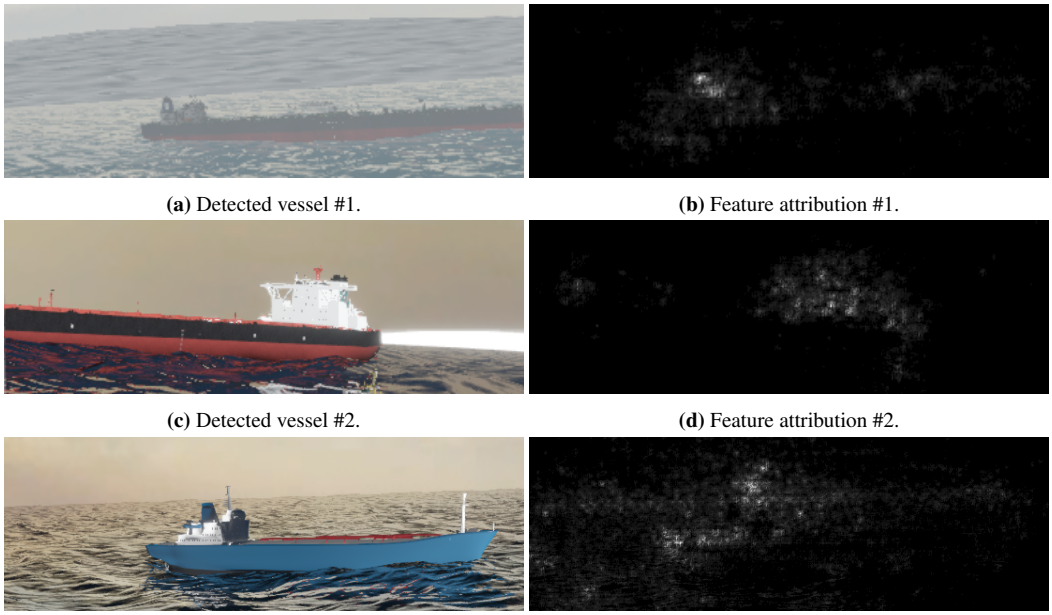
(c) Detected vessel #2.

(d) Feature attribution #2.

(e) Detected vessel #3.

(f) Feature attribution #3.

Figure 7.18: Examples of three non-intuitive attributions from the LIME based feature attribution method.



(a) Detected vessel #1. **(b)** Feature attribution #1.

(c) Detected vessel #2. **(d)** Feature attribution #2.

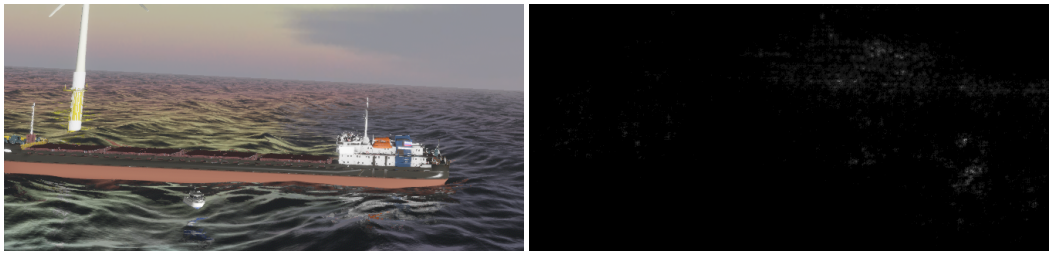
(e) Detected vessel #3. **(f)** Feature attribution #3.

Figure 7.19: Examples of the Integrated Gradients based method highlighting the superstructures of detected vessels. The method was configured to use the parallel decomposition merging function.

features in its predictions. Figure 7.19 highlights a few examples of this behaviour. This effect seems to strengthen when the vessel is partly occluded. This can be seen from the figure, as the two instances of partly occluded ships exhibits less noisy feature attributions. This could indicate that the model relies more heavily on the superstructure when denied access to other features of the vessel. This would mean that the model is somewhat robust to partly occluded ships, which are frequent in the synthetic dataset. It is worth noting that in some cases, even some in which the superstructure is clearly visible, the feature attribution method does not highlight it. It is unclear whether this is because the Mask RCNN model does not utilize the superstructure in these specific predictions or if the feature attribution method just fails to highlight it. Figure 7.20 shows an example of this. Sometimes the network will highlight the position where a superstructure would have been if the ship was oriented in the opposite direction. Figure 7.21 highlights this behaviour. A possible reason for this could be that the network highlights the lack of superstructure as an important feature.

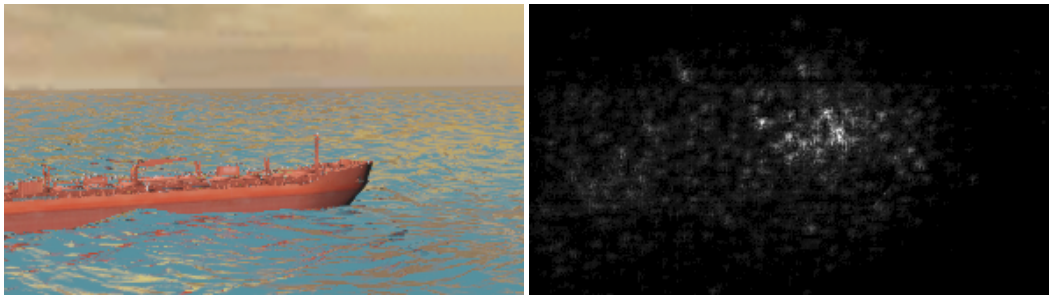
Some of the insights discussed so far might seem vague. This is due to an inherit limitation of feature attribution methods such as the two employed in this experiment. Since the methods only highlights parts of the input image, but do not express anything about the model's actual decision process, the user is left to deduce this on his/her own. It is therefore dangerous to draw too many conclusions from these results, as the logic used to deduce them might not match the realities of the actual Mask RCNN model.

Other features that often appeared in the attributions are the vessel's bow, the aft of the vessel,



(a) Detected vessel. **(b)** Feature attribution.

Figure 7.20: For some detections the feature attributions do not include the superstructure. It is unknown whether this is because the Mask RCNN model does not use it in its predictions or because of a failure in the feature attribution system. The method used the parallel decomposition merging function.

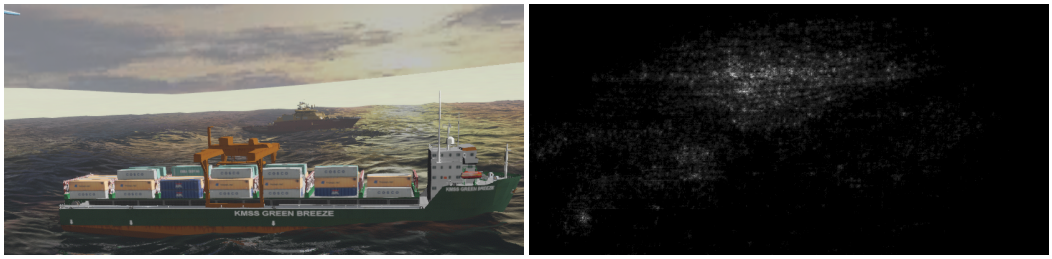


(a) Detected vessel. **(b)** Feature attribution.

Figure 7.21: In some cases the method highlights areas of the image where a superstructure would have been had the ship been oriented in the opposite direction. This could indicate that the network uses the lack of superstructure in its predictions.

and the waterline close to the ends of the ship. As with the vessel's superstructure these are relatively complex features, and it is reasonable to expect that the network has learned to represent them in the final layers of the feature maps.

In the samples tested, the Integrated Gradients based method frequently highlighted the background terrain in its feature attributions. This effect was especially strong if the terrain was white, which is meant to emulate snow. Figure 7.22 shows a feature attribution suffering from this effect. There could be several reasons for this. The Mask RCNN model could be using the horizon in its predictions, and hence the background terrain would influence the predictions by changing the geometry of the horizon. Another theory is that the white color of the terrain looks like a vessel's superstructure and confuses the model. After all, the feature attributions do indicate that the model often bases its predictions on existing superstructure, if it is visible in the image. If the issue is systemic in the Mask RCNN model and not an artifact of the feature attribution, the model should be trained on more images with background terrain. This could be done by implementing a more sophisticated terrain generator in the synthetic dataset generation software, or by gathering real world data with terrain visible. Creating more synthetic images might also reduce the issue.



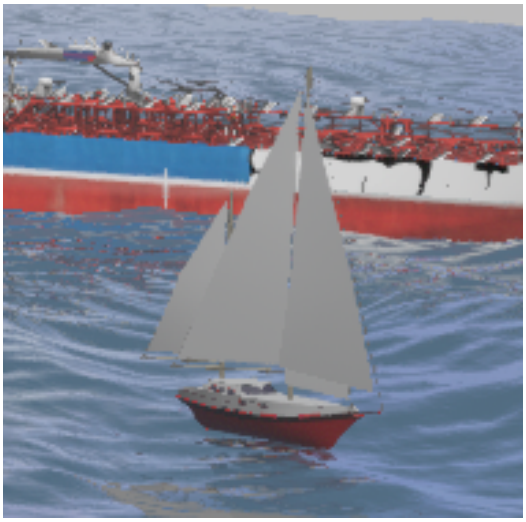
(a) Detected vessel. **(b)** Feature attribution.

Figure 7.22: Example of feature attribution noise caused by background terrain.

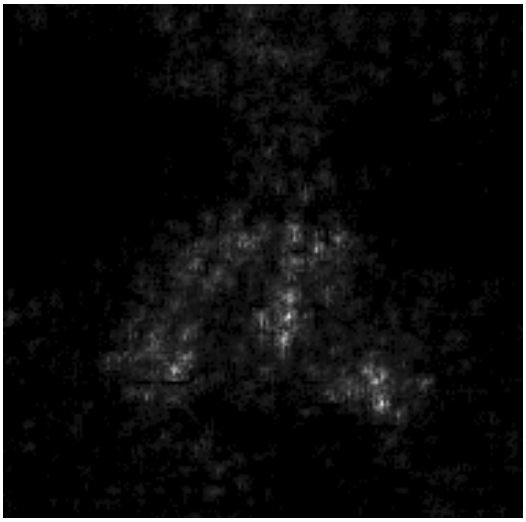
As noted earlier in Section 7.1.1, the Mask RCNN networks are suspected to be biased with regards to the sail geometry of the sailboat class because of their unreasonably high performance when tested on this class. This suspicion could be examined through feature attribution - if the networks are biased with respect to the sail geometry, the sail would be highlighted more clearly than the ship's hull in the feature attributions. Figure 7.23 highlights two feature attributions on the sailboat class generated by the Integrated Gradients based method. The attributions do indeed seem to indicate that the network relies heavily on the sail, since in most feature attributions only the sail is highlighted as an important feature. To clarify: this is a problem because in the real world, the sail geometry might vary between boats and change due to weather conditions. The sailboat's actual heading is defined by its hull. This problem could be alleviated by including more varied 3D models depicting sailboats, with different sail configurations.

It was found that if the detected object was represented by a small section of the image, the quality of the feature attribution often suffered, simply because there were not enough pixels to facilitate an accurate feature attribution. Because some defined classes are smaller than other, for instance the motorboat class, detections of these classes frequently led to feature attributions of poor quality. The attributions were often limited to just highlighting the entire vessel - this is not completely useless information, at least it assures the user that the network is actually evaluating the correct area of the image. Still, a more detailed explanation of the prediction is desired. Figure 7.24 highlights a few such examples. The cropped analysis mode was intended to somewhat alleviate this issue, but the fundamentally poor performance of the LIME based model itself prevented detailed analysis of the potential improvements offered by the cropping modification. This could be examined more closely in a future experiment. To fully alleviate this issue it is likely that a different method of XAI would be needed, as the quality of feature attributions are too dependant on the size of the detected objects in the image frame.

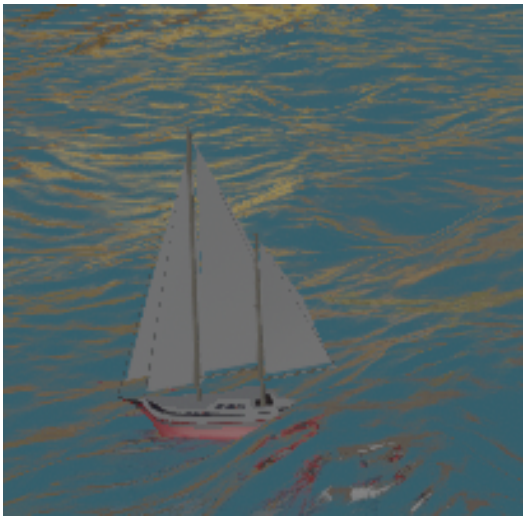
So why does the two feature attribution methods perform worse on the maritime dataset? There are likely several factors at play, but the main theory is that the increased complexity of the prediction process within the Mask RCNN network made the feature attribution problem more difficult. Recall the "Blob World" dataset from the previous section. In it, the blobs' headings were always defined by a single, clearly visible, feature. This likely caused the internal prediction process of the Mask RCNN model trained on the blob dataset to be much simpler than the one trained on the



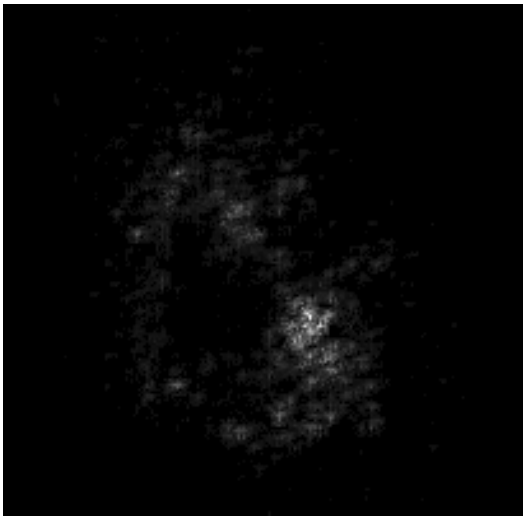
(a) Detected vessel #1.



(b) Feature attribution #1.

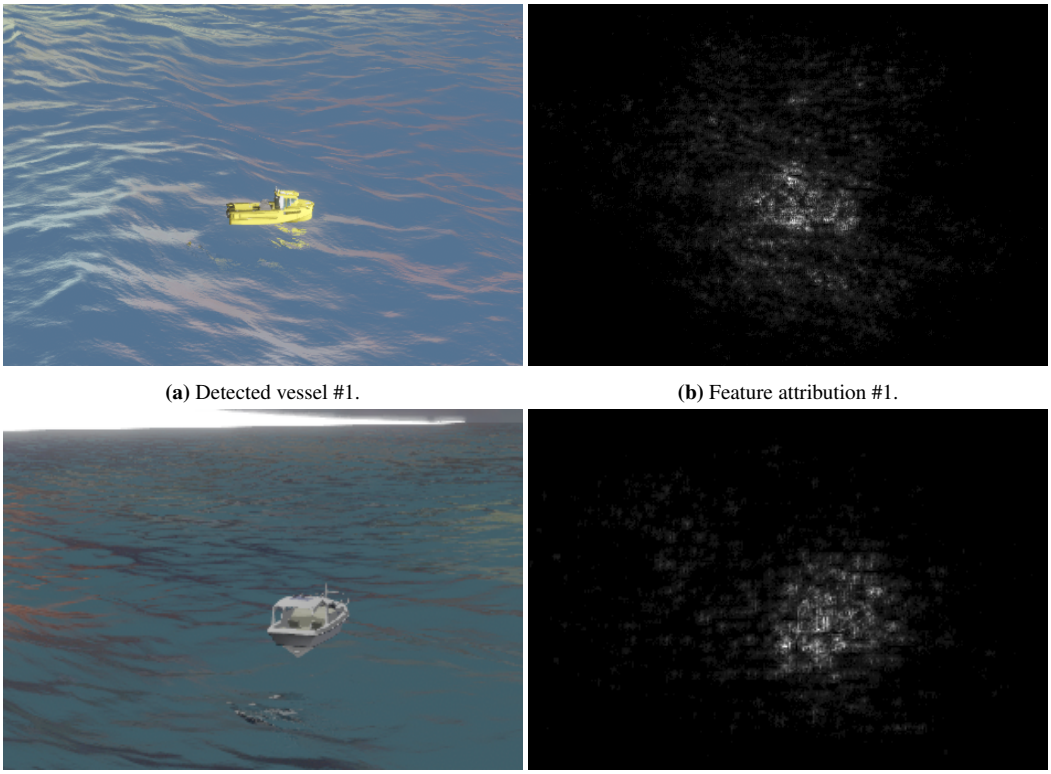


(c) Detected vessel #2.



(d) Feature attribution #2.

Figure 7.23: Feature attributions on the sailboat class. Observe how the network primarily highlights the sail. This could indicate that the network suffers from a bias.



(a) Detected vessel #1.

(b) Feature attribution #1.

(c) Detected vessel #2.

(d) Feature attribution #2.

Figure 7.24: Objects that appear small in the images often lead to non-precise feature attributions.

maritime dataset. As an example, in the maritime dataset, just detecting a vessel's superstructure is not enough. It must be detected in relation to other features present on the ship. Otherwise the model would not be able to accurately predict the heading of both supply ships and tankers, since their headings are defined differently with regards to the position of the ship's superstructure. There are also other vessels defined in the maritime dataset that lack a clear superstructure, so the model must learn to base its predictions on a myriad of features. It is well-known that the LIME feature attribution method can struggle with highly nonlinear relationships since the linear surrogate model it uses internally fails to represent the local behaviour of the system accurately. Since one of the feature attributions methods are based on the LIME method, this could be one reason for its poor performance on this dataset.

The blobs in the 'Blob World' dataset were always clearly visible, single colored, and lived on a uniform black background. This simplicity allowed the Mask RCNN model trained on this dataset to reach very high performance. On the other hand, the model trained on the maritime dataset did not reach the same high performance. The lower performance might also be because the model is influenced by image features not relevant to the heading of the detected vessel. In the feature attributions, this would be visible as noise. This implies that some of the noise visible in the feature attributions on the maritime dataset might actually stem from the Mask RCNN model itself, and is not caused by the feature attribution methods. However, how much noise comes from the Mask RCNN model and how much comes from the feature attribution is not known. In the future, a study could examine this by performing feature attributions on a model in various parts of its training process. Examining how the noise in the feature attribution changes during the training process could give an indication of how much of it stems from inaccuracies in the Mask RCNN model and how much stems from the feature attribution system.

It is also possible that the LIME based method performed poorly because it needs to be tuned to each specific dataset to be able to generate accurate feature attributions. This is discussed further in Section 7.2.1. This experiment did not fine-tune the LIME based method to fit the maritime dataset, instead opting to use the configuration that worked best on the 'Blob World' dataset. It is possible that more insight into the Mask RCNN model's decision process could be achieved if the LIME based method was better tuned for this dataset.

7.3 Including Pixel-Accurate Depth Information

This section presents the results of the experiment detailed in Section 6.4. The models' validation loss during the training process is presented, as well as their performance with respect to both object detection and heading estimation. Let's begin with the validation loss during the models' training progress. It is shown in Figure 7.26. The baseline model, which does not have access to depth information, is shown in black, the simple-fuse model is shown in blue, and the deep-fuse model is shown in green. From the figure it is clear that all three models converge to a comparable performance level, at least with regards to the validation loss. It can also be seen that the deep-fuse model needs more time to converge to the same level. This is intuitive, since its network architecture has undergone the most change, which the network's pretrained parameters must adapt to. The network also has to learn the 1x1 convolutional filters used to merge the depth-maps with the feature maps from scratch.

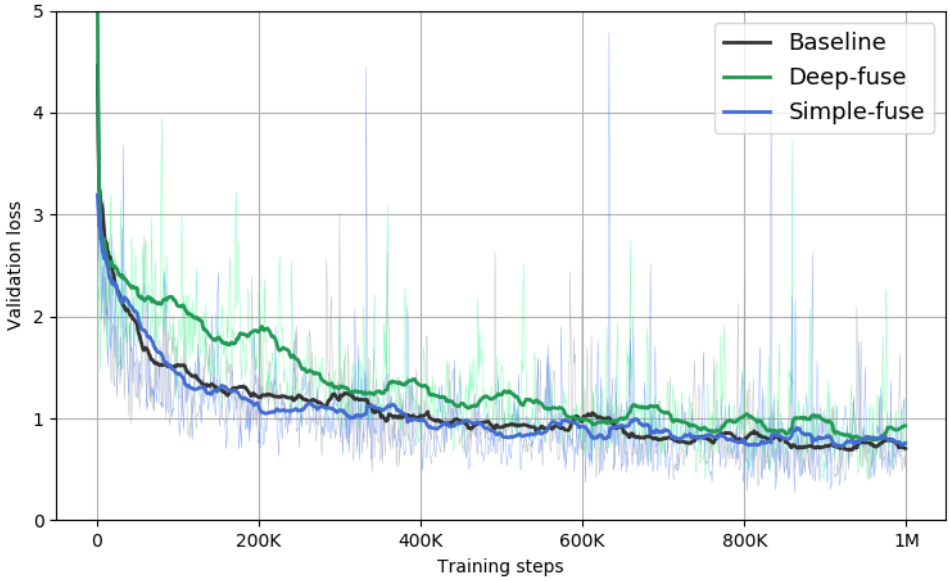


Figure 7.26: The validation loss of the three models during training. The plots have been smoothed with a rolling average filter (N=50).

It is also interesting that the simple-fuse model follows almost the exact same trajectory as the baseline model, this could indicate that the network's parameters adapt quickly to the inclusion of the depth-maps in the fourth image channel.

The models' mAP scores are shown in Figure 7.25. The scores indicate that the simple-fuse model performs best with regards to the object detection task. Disappointingly, the figure also shows that the deep-fuse model performs worse than the baseline model. This indicates that it has not learned to use the additional depth information as effectively as the simple-fuse model. In fact, its inclusion degrades the model's performance. When compared to the Mask RCNN models trained in the first experiment, see Figure 7.7 in Section 7.1.5, the simple-fuse model performs better than any previously tested Mask RCNN model with a heading estimation module. When comparing the baseline models in the two experiments, one can observe the degrading effect of not being allowed to use a pretrained first layer in the feature extractor, as that is the only difference between the two baseline models. The difference of 7.26% is quite remarkable.

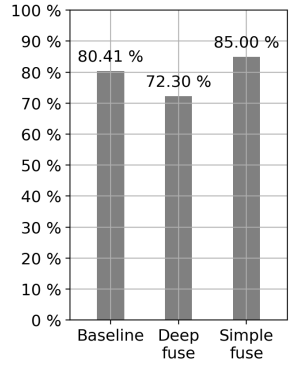


Figure 7.25: The mAP scores achieved by the models when tested on the validation dataset of 2000 images.

Model	MeanAE	MedianAE
Baseline	36.91°	11.18°
Deep-fuse	46.94°	17.60°
Simple-fuse	31.67°	9.97°

Table 7.2: The heading estimation performance the three models achieved when tested on the validation dataset of 2000 images.

Table 7.2 details the models' heading predictions meanAE and medianAE. As in the earlier performance tests, the simple-fuse model performs the best, while the deep-fuse model continues to perform poorly. Comparing these results with the ones achieved with the earlier heading estimation models, see Table 7.1 and Figure 7.3, the simple fuse model is the second highest performing model, only beaten by the classification-based model predicting the apparent heading. When factoring in the errors caused by the transformation from apparent heading to true heading it might even become the best performing model. It is however, very interesting that the models with access to depth information are unable to perform much better than the models restricted to only using the visual data.

So why do the models fail to improve as much as one might expect them to? Well, it is likely down to a number of reasons. Let's focus on the deep-fuse model as it performed the worst. As noted in Section 6.4, the deep-fuse technique implemented in this work is a modified version of the one presented in [21]. In their work the researchers are able to demonstrate a performance increase, albeit small, over their baseline model. Maybe the differences between their architecture and the one proposed in this work leads to the degraded performance? The deep-fuse model implemented in this thesis added the depth-maps to the feature maps after a 1x1 convolutional layer. This was done to preserve the dimensions of the feature maps but might also have degraded the model's ability to create clear depth based features. The original architecture concatenated the depth-maps with the feature maps, which might have preserved the depth based features better. The depth-maps used in this experiment are also different from the ones used in the original architecture, as they are dense compared to the sparse converted radar measurements used in [21].

Another possible cause for the lack of improvement is the extensive use of pretrained parameters in the models trained. These pretrained parameters were generated on the COCO dataset, which does not contain samples with depth information. It is possible that this has trapped the network's performance in a local optima, preventing it from correctly utilizing the depth information. The original architecture presented by the researchers at the Technical University of Munich also used pretrained weights for their VGG feature extractor[21], so the severity of this effect is unknown. In the future, another experiment could train the networks without using pretrained weights. If a performance gap appears this could indicate that the networks were indeed stuck in a local optima.

The depth-maps generated for this experiment were limited to a maximum distance of 2000 meters. A larger detection range than this was deemed unnecessary, since the maximum vessel distance from the camera was set to 1000 meters. See Figure 7.5 in Section 7.1.3 for the vessel distribution with respect to distance. However, this limit meant that sometimes parts of the background terrain would not be visible in the depth-maps, due to its distance from the camera. It is however unlikely

that this would decrease the models' performances, since the models were already very good at detecting terrain without access to depth-maps and since all the vessels are placed within the maximum range of the depth-maps and are clearly visible. Finally, another reason for the lack of performance might simply be a lack of training. Since the deep-fuse model's performance converges slower than the other two models, as shown in Figure 7.26, it might just require more training time before it learns to utilize the depth information in its predictions. When this theory was put to the test, by training a deep-fuse model for 2000K training steps, the model failed to improve much. It is therefore unlikely that the lack of performance is due to a lack of training time.

Chapter 8

Conclusion and Further Work

This chapter aims to conclude the experiments on the Mask RCNN architecture and synthetic dataset generation, and suggest experiments to be performed in further works. Since the synthetic dataset generation system developed in this thesis can be used for practically any computer vision experiment, the proposed experiments will be limited to a select few. This chapter will begin with a recap of the experiments performed. Ideas for further work will be presented intermittently.

The system for synthetic dataset generation was further developed and used in all three experiments. New features include pixel accurate depth-maps, more robust vessel placement, and the inclusion of more information about the objects in the images. In future works, these added features would allow simulation of various depth sensing sensor systems, as well as experiments with other prediction tasks, such as depth estimation from monocular images. The synthetic dataset generation system itself could also be improved. Especially the terrain generation system could be modified to better represent real world terrain topologies. To conclude the work on synthetic dataset generation: after the work done during this master thesis and the proceeding project thesis, the synthetic dataset generation system performs well and is suited for a wide selection of computer vision experiments.

The first experiment evaluated three modified versions of the Mask RCNN architecture for estimating the heading of detected objects. One estimated the heading as a single heading vector, one estimated a heading vector per defined class and later used the one corresponding to the classification result, and one reformulated the problem as a set of classification problems and generated the final prediction through a mean shift clustering algorithm. All three methods proved to be reasonably resilient to changes in distance, vessel orientation, and vessel type. Of the three methods, the classification-based method achieved the highest performance. When compared to humans, all three methods performed within the bounds of measured human performance. The networks trained in this thesis were initialized with parameters pretrained on the COCO dataset. These pretrained parameters had never performed heading estimation since the COCO dataset does not include heading information. Therefore, some amount of transfer learning was required. This might have reduced the final performance of the models. In a future experiment one could train these networks from scratch on the synthetic datasets. In that case, a larger dataset would likely be required, as well as more computing power to facilitate faster training and a larger batch size. It would also be interesting to see whether the methods could be adapted to work on dark images, where the models would be restricted to using the vessels' navigational lights in their predictions. To facilitate this, the synthetic dataset generation software would have to be modified to create night-time images and to include navigational lights. Another interesting question is how well the methods would perform on real world datasets. Naturally, extensive domain adaption would likely be required before their performance on a real world dataset would match the ones achieved on the purely synthetic dataset. To do this experiment, one would first have to collect a real world dataset, which might be a time consuming process. To conclude this experiment: all three methods for heading estimation performed well, and their modular design makes them simple to add to other network architectures. This makes them suitable for further experimentation or use in products.

The second experiment modified two feature attribution algorithms to explain the predictions made by the heading estimation systems developed in the first experiment. The two methods were based on the Integrated Gradients method and the LIME method. A simple toy dataset was used to validate the two models, on which they performed fine. When tasked with generating feature attributions for the Mask RCNN model trained on the more complex synthetic maritime dataset, they both struggled. The Integrated Gradients based method performed best, but the attributions were generally noisy and sometimes non-intuitive. The LIME based method generally failed to highlight any intuitive features. It became evident during the development of these methods that the best way to validate their functionality was by testing them on very simple datasets. In a future experiment the 'Blob World' dataset could be expanded upon, for example by including false features which do

not influence the heading of the objects. A well-performing feature attribution method should then ignore these false features and correctly highlight the ones that actually influence the heading of the objects. The two feature attribution methods were highly configurable, and a future experiment could test more configurations. Filtering techniques to reduce the noise in the attributions could also be experimented with. The LIME based method implemented in this thesis was modified with a cropped analysis feature. A future work could experiment further with this feature to determine its potential benefits. To conclude this experiment: the two feature attribution methods certainly provide some value to developers, but their benefit to the end user is uncertain. One also has to be keenly aware one's confirmation bias, as it is always possible to find an attribution to fit any theory one might have about the computer vision system's internal process, but when used wisely they can certainly be useful.

The third experiment analysed two ways of including depth information in the Mask RCNN architecture's input. The first method simply added the normalized depth-maps as a fourth image channel along with the preexisting RGB channels. The second method is inspired by a learnable architecture for fusing radar measurements internally in the feature extractors of neural networks. It worked by gradually decreasing the size of the depth-maps through average pooling and adding them to the various levels of the feature pyramid after a 1x1 convolutional operation. When compared to a baseline model the first method performed better, while the second method performed worse than the baseline model. Several reasons for this result were hypothesised in Section 7.3, and the true reason is likely a mix of them. In the future, a more detailed analysis of depth inclusion could be performed without the use of pretrained network parameters. This experiment would require more computing power to facilitate quicker training and a larger batch size. To conclude this experiment: the proposed method for a learnable fusion of depth information in the Mask RCNN feature extractor failed to yield a performance increase. It is possible that through more experiments and testing a modified method might achieve better performance.

It is time to finish things up and look at the bigger picture. This thesis has demonstrated three experiments using synthetic datasets. It has implemented modified versions of the Mask RCNN architecture which estimate the heading of vessels within a maritime scene. As discussed in the introduction, such a system could be part of an autonomous vessel's sensory system, which again would be a part of its situational awareness system. This way the systems developed in this thesis' first experiment could be viewed as an enabling technology for autonomous vessels. The second experiment in this thesis attempted to use two modified feature attribution methods to gain insight into the prediction process of Mask RCNN networks. The field of modern XAI is still in its early days and the methods used in this experiment might seem crude in a few years time, but at the moment they can still serve as valuable tools for developers by allowing them insight into their models. The third experiment tested methods for including pixel accurate depth-maps in a Mask RCNN model's input. Currently no sensor system can produce such accurate depth-maps of an environment, and the experiment was mostly meant to demonstrate the newly developed functionality of the synthetic dataset generation system. Sensor fusion is already common in autonomous vehicle operations, and the synthetic dataset generation system's functionality demonstrated in this experiment allows new experiments on combining data from depth sensors with data from image sensors. It is clear that all three experiments in this thesis serve to further the dream of fully autonomous maritime vehicles. Let's hope that this dream once becomes reality.

Bibliography

[1] L. M. Angelsen, "Explainability of Instance Segmentation Models Trained on Synthetic Datasets." Project thesis, NTNU, 2019.

[2] W. Abdulla, "Mask R-Cnn for Object Detection and Instance Segmentation on Keras and Tensorflow."
`https://www.github.com/matterport/Mask_RCNN/`, 2017.

[3] K. Dhamdhere, P. K. Mudrakarta, M. Sundararajan, A. Taly, and J. Xu, "Integrated Gradients."
`https://www.github.com/ankurtaly/Integrated-Gradients/`, 2016.

[4] M. T. C. Ribeiro, "Lime: Explaining the Predictions of Any Machine Learning Classifier."
`https://www.github.com/marcotcr/lime/`, 2017.

[5] R. Padilla, "Object-Detection-Metrics."
`https://github.com/rafaelpadilla/Object-Detection-Metrics/`, 2018.

[6] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[7] 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*, pp. 580–587, 2014.

[8] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, pp. 91–99, 2015.

[9] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, pp. 2961–2969, 2017.

[10] 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*, pp. 770–778, 2016.

[11] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440, 2015.

[12] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241, Springer, 2015.

[13] 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*, pp. 2117–2125, 2017.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[15] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International journal of computer vision*, vol. 88, no. 2, pp. 303–338, 2010.

[16] Adrian Rosebrock, "Intersection over Union (IoU) for object detection." `https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/`. Accessed: 26.2.2019.

[17] J. Tremblay, T. To, and S. Birchfield, "Falling things: A synthetic dataset for 3d object detection and pose estimation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 2038–2041, 2018.

[18] N. Heess, D. TB, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. Eslami, *et al.*, "Emergence of locomotion behaviours in rich environments,"

[19] S. Gustavson, "Simplex noise demystified," *Linköping University, Linköping, Sweden, Research Report*, 2005.

[20] "Colorizer website." `https://www.colorizer.org`. Accessed: 7.10.2019.

[21] F. Nobis, M. Geisslinger, M. Weber, J. Betz, and M. Lienkamp, "A deep learning-based radar and camera sensor fusion architecture for object detection," in *2019 Sensor Data Fusion: Trends, Solutions, Applications (SDF)*, pp. 1–7, IEEE, 2019.

[22] T. I. Fossen, *Handbook of marine craft hydrodynamics and motion control*. John Wiley & Sons, 2011.

[23] "Tips for trygg sjøferd i mørket." `https://www.redningsselskapet.no/sikker-til-sjos/lanterneforing-i-morket/`. Accessed: 3.9.2020.

[24] F. Chabot, M. Chaouch, J. Rabarisoa, C. Teuliere, and T. Chateau, "Deep manta: A coarse-to-fine many-task network for joint 2d and 3d vehicle analysis from monocular image," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2040–2049, 2017.

[25] K. Hara, R. Vemulapalli, and R. Chellappa, "Designing Deep Convolutional Neural Networks for Continuous Object Orientation Estimation," 2017.

[26] T. Miller, "Explanation in artificial intelligence: Insights from the social sciences," *Artificial Intelligence*, vol. 267, pp. 1–38, 2019.

[27] M. T. Ribeiro, S. Singh, and C. Guestrin, "" why should i trust you?" explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1135–1144, 2016.

[28] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[29] A. Krebs, A. Lenci, and D. Paperno, "Semeval-2018 task 10: Capturing discriminative attributes," in *Proceedings of The 12th International Workshop on Semantic Evaluation*, pp. 732–740, 2018.

[30] A. Wan, L. Dunlap, D. Ho, J. Yin, S. Lee, H. Jin, S. Petryk, S. A. Bargal, and J. E. Gonzalez, "Nbdt: Neural-backed decision trees," 2020.

[31] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," *preprint*, 12 2013.

[32] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic attribution for deep networks," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 3319–3328, JMLR. org, 2017.

[33] K. Hara, R. Vemulapalli, and R. Chellappa, "Designing deep convolutional neural networks for continuous object orientation estimation," 02 2017.

[34] Andrej Karpathy, "AI for Full-Self Driving." `https://youtu.be/hx7BXih7zx8`. Accessed: 28.04.2020.

[35] "Initial sea trials successfully completed by Wartsila & PSA Marine's ground-breaking 'IntelliTug' project." `https://www.wartsila.com/media/news/13-03-2020-initial-sea-trials-successfully-completed-by-wartsila-psa-marine-s-ground-breaking-intellitug-project-2660851`. Accessed: 6.11.2020.

[36] "Verdens første helt autonome fergeseilas gjennomført - teknologien er 100 prosent klar." `https://www.tu.no/artikler/verdens-forste-helt-autonome-fergeseilas-gjennomfort-teknologien-er-100-prosent-klar/452610`. Accessed: 6.11.2020.

[37] Darrell Etherington, "Waymo Has Now Driven 10 Billion Autonomous Miles in Simulation." `https://www.techcrunch.com/2019/07/10/waymo-has-now-driven-10-billion-autonomous-miles-in-simulation/`. Accessed: 2019-11-14.

[38] A. T. Henriksen, "Domain Adaptation for Maritime Instance Segmentation: From Synthetic Data to the Real-World," Master's thesis, NTNU, 2019.

[39] V. Bolón-Canedo, N. Sánchez-Maroño, and A. Alonso-Betanzos, "A review of feature selection methods on synthetic data," *Knowledge and information systems*, vol. 34, no. 3, pp. 483–519, 2013.

[40] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

[41] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of Machine Learning*. MIT Associated Press, second ed., 2018.

[42] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[43] Michael Nielsen, "Neural Networks and Deep Learning." `http://neuralnetworksanddeeplearning.com/`. Accessed: 29.08.2019.

[44] F. Rosenblatt, *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.

[45] M. Minsky and S. Papert, *Perceptrons*. MIT Press, 1969.

[46] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[47] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function," *Neural networks*, vol. 6, no. 6, pp. 861–867, 1993.

[48] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, vol. 30, p. 3, 2013.

[49] Piotr Skalski, "Gentle Dive into Math Behind Convolutional Neural Networks Part 5." `https://www.towardsdatascience.com/gentle-dive-into-math-behind-convolutional-neural-networks-79a07dd44cf9/`. Accessed: 30.08.2019.

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

[51] J. Uijlings, K. van de Sande, T. Gevers, and A. Smeulders, "Selective Search for Object Recognition," *International Journal of Computer Vision vol. 104*, 2012.

[52] H. H. Tan and K. H. Lim, "Review of second-order optimization techniques in artificial neural networks backpropagation," in *IOP Conference Series: Materials Science and Engineering*, vol. 495, p. 012003, IOP Publishing, 2019.

[53] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*, pp. 177–186, Springer, 2010.

[54] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural networks*, vol. 12, no. 1, pp. 145–151, 1999.

[55] T. Jayalakshmi and A. Santhakumaran, "Statistical normalization and back propagation for classification," *International Journal of Computer Theory and Engineering*, vol. 3, no. 1, pp. 1793–8201, 2011.

[56] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015.

[57] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, "How does batch normalization help optimization?," in *Advances in Neural Information Processing Systems*, pp. 2483–2493, 2018.

[58] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, *et al.*, "Language models are few-shot learners,"

[59] "A Gentle Introduction to Dropout for Regularizing Deep Neural Networks." `https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/`. Accessed: 2.25.2019.

[60] X. Zheng, T. Chalasani, K. Ghosal, S. Lutz, and A. Smolic, "Stada: Style transfer as data augmentation," 2019.

[61] P. T. Jackson, A. Atapour-Abarghouei, S. Bonner, T. Breckon, and B. Obara, "Style augmentation: Data augmentation via style randomization," pp. 1–13, 2018.

[62] "Common Objects in Context Website." `http://www.cocodataset.org/#detection-eval/`. Accessed: 03.09.2019.

[63] D. Dwibedi, I. Misra, and M. Hebert, "Cut, paste and learn: Surprisingly easy synthesis for instance detection," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1301–1310, 2017.

[64] I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, *et al.*, "Solving rubik's cube with a robot hand,"

[65] "Norwegian Coastal Administration Rules On AIS use." `https://www.kystverket.no/Maritime-tjenester/Meldings--og-informasjonstjenester/AIS/AIS-regelverk-og-brukarkrav/`. Accessed: 3.9.2020.

[66] M. Gabb, O. Löhlein, M. Oberländer, and G. Heidemann, "Efficient monocular vehicle orientation estimation using a tree-based classifier," in *2011 IEEE Intelligent Vehicles Symposium (IV)*, pp. 308–313, IEEE, 2011.

[67] J. Choi, B.-J. Lee, and B.-T. Zhang, "Human body orientation estimation using convolutional neural network," 2016.

[68] Y. Xiang, W. Kim, W. Chen, J. Ji, C. Choy, H. Su, R. Mottaghi, L. Guibas, and S. Savarese, "Objectnet3d: A large scale database for 3d object recognition," in *European Conference on Computer Vision*, pp. 160–176, Springer, 2016.

[69] Y. Wang, X. Tan, Y. Yang, X. Liu, E. Ding, F. Zhou, and L. S. Davis, "3d pose estimation for fine-grained object categories," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[70] Y. Cheng, "Mean shift, mode seeking, and clustering," *IEEE transactions on pattern analysis and machine intelligence*, vol. 17, no. 8, pp. 790–799, 1995.

[71] M. Van Lent, W. Fisher, and M. Mancuso, "An explainable artificial intelligence system for small-unit tactical behavior," in *Proceedings of the national conference on artificial intelligence*, pp. 900–907, Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2004.

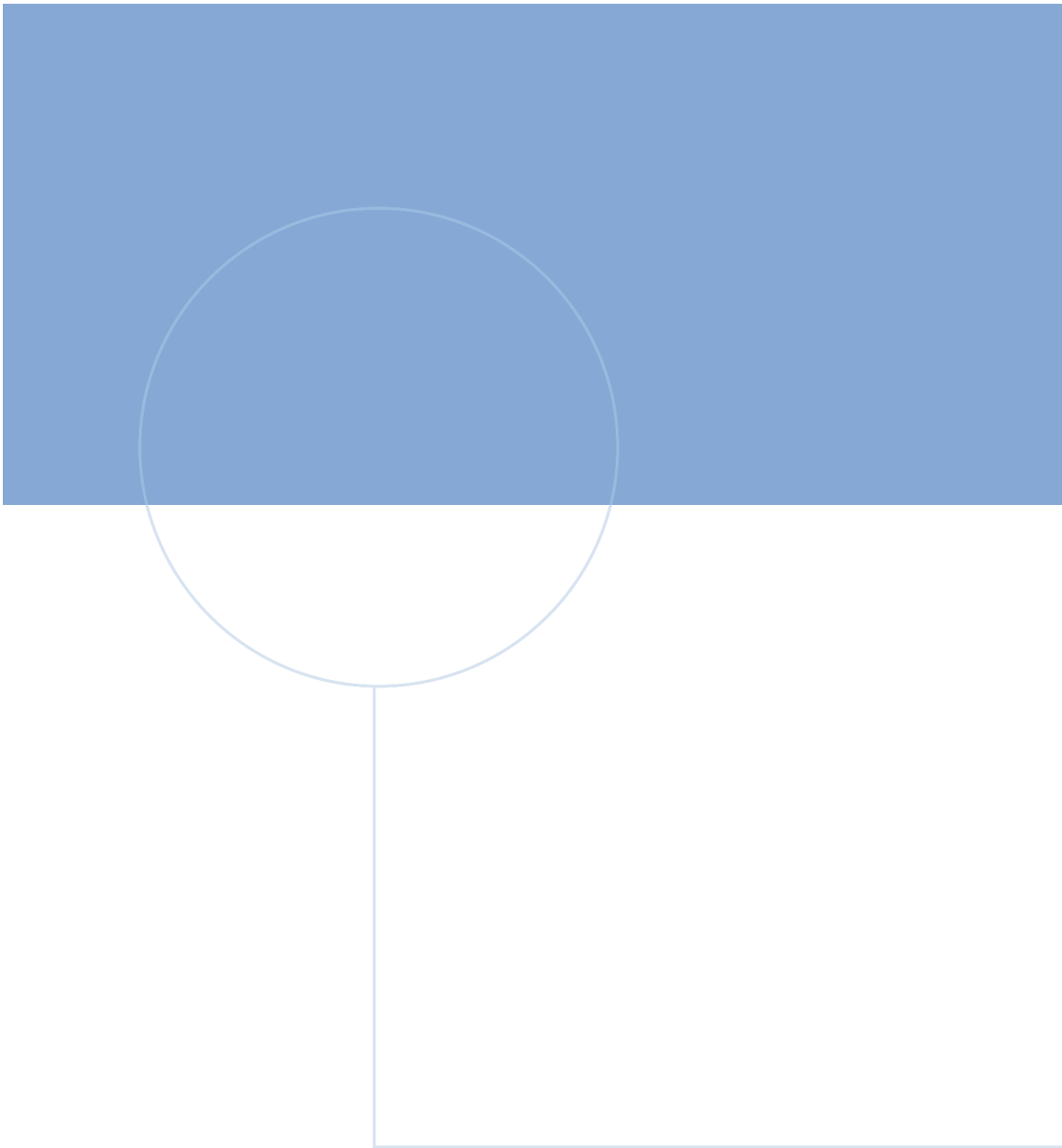
[72] F. Doshi-Velez and B. Kim, "Towards a rigorous science of interpretable machine learning," 2017.

[73] A. Adadi and M. Berrada, "Peeking inside the black-box: A survey on explainable artificial intelligence (xai)," *IEEE Access*, vol. 6, pp. 52138–52160, 2018.

[74] D. Kahneman and A. Tversky, "The simulation heuristic.," tech. rep., Stanford Univ CA Dept of Psychology, 1981.

[75] A. Stepanjans and A. Freitas, "Identifying and explaining discriminative attributes," 2019.

[76] "EPFL Website." `https://www.epfl.ch/labs/cvlab/data/data-pose-index-php/`. Accessed: 1.4.2020.



NTNU

Norwegian University of
Science and Technology