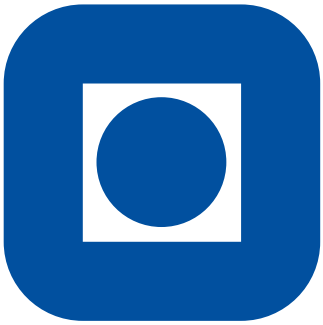
Lars Mansåker Angelsen

Explainability of Instance Segmentation Models Trained on Synthetic Datasets

Applying Explainable AI Methods to a Mask RCNN Model Trained on Synthetic Data Generated by the Kongsberg Cogs Graphics Engine



NTNU

Supervisor:
Anastasios Lekkas

Project Thesis
Department of Engineering Cybernetics
Norwegian University of Science and Technology
Norway
December 2019

Summary

This thesis explores how explainable artificial intelligence can be used to gain insights on the predictions made by an instance segmentation algorithm. This search is motivated by the expected near term adaption of computer vision algorithms like instance segmentation in autonomous vehicles. This safety critical area requires a clear understanding of the artificial intelligence algorithms used. Furthermore, explainable artificial intelligence can be used as a tool for improving the performance of an artificial intelligence system by shedding light on a system's weaknesses.

A procedure for generating synthetic datasets using Kongsberg Cogs is developed. Currently the datasets used to train artificial intelligence systems are often collected and labelled manually. This is a time consuming process and synthetic dataset generation is poised to be a much quicker alternative. Kongsberg Cogs is interfaced with via a Python script which facilitates dynamic model loading and 3D scene configuration. The developed system uses 150 3D models of ships and can generate 1000 images per hour without requiring human labor.

A generated dataset consisting of 23500 unique images is then used to train an instance segmentation algorithm called Mask RCNN. The instance segmentation task consists of detecting the objects within a scene, classifying them, and generating a mask covering their silhouettes. An experiment where the instance segmentation algorithm is tested on a few synthetic and real world images is performed. The resulting model achieves a $mAP = 0.8937$ when using an Intersection Over Union (IOU) threshold equal to 0.5, when tested on a synthetic dataset. When applied to real world images the model performs worse, but still delivers acceptable results in many cases.

Finally, three explainable artificial intelligence methods are tasked with generating explanations for a few detections. The three methods are: examining the Jacobian matrix, the integrated gradients algorithm, and the Local Interpretable Model-agnostic Explanations (LIME) algorithm. These algorithms are then compared to each other. As it turns out, examining the Jacobian matrix does not offer much insight into the model. The integrated gradients method performs better and the LIME method performs the best.

Sammendrag

Denne oppgaven utforsker hvordan forklarbar kunstig intelligens kan brukes til oppnå innsikt i prediksjonene fra en instans segmenterings algoritme. Dette er motivert av den forventede bruken av datasyn algoritmer som instans segmenterings algoritmer i autonome fartøy i nær fremtid. I denne sikkerhetskritiske applikasjonen er det nødvendig med en klar forståelse av de anvente kunstig intelligens algoritmene. Forklarbar kunstig intelligens kan også brukes som et verktøy for å forbedre et kunstig intelligens system ved å belyse systemets svakheter.

En prosedyre for å generere kunstige datasett ved hjelp av Kongsberg Cogs er utviklet. Slik det er nå blir ofte datasett samlet og merket manuelt. Dette er en treg prosess og syntetisk datasett generasjon ligger an til vre et mye raskere alternativ. Kongsberg Cogs blir koblet til via et Python script, dette muliggjør dynamisk model lasting og 3D scene generasjon. Det endelige systemet bruker 150 3D modeller og kan generere 1000 bilder i timen uten at menneskelig arbeid er nødvendig.

Et datasett bestående av 23500 unike bilder blir så brukt til å trene en instans-segmentering algoritme kalt Mask RCNN. Instans-segmentering oppgaven innebærer å detektere objektene i en scene, klassifisere dem, og generere en maske som dekker objektenes silhuetter. Et eksperiment hvor instans-segmentering algoritmen blir testet på et par syntetiske og ekte bilder blir utført. Den resulterende modellen oppnår en $mAP = 0.8937$ med en Intersection Over Union (IOU) terskel tilsvarende 0.5, når den testes på det syntetiske datasettet. Når den brukes på ekte bilder yter den dårligere, men leverer fortsatt et akseptabelt resultat i mange tilfeller.

Til slutt brukes tre forklarbar kunstig intelligens metoder til å generere forklaringer for en håndfull deteksjoner. De tre metodene er: eksaminasjon av Jacobi-matrisen, integrated gradients algoritmen, og Local Interpretable Model-agnostic Explanations (LIME) algoritmen. Disse algoritmene blir så sammenlignet med hverandre. Resultatet er at eksaminasjon av Jacobi-matrisen ikke gir mye innsikt i modellen. Integrated gradients algoritmen gir bedre resultater og Local Interpretable Model-agnostic Explanations (LIME) gir de beste resultatene.

Preface

This thesis was written in my second to last semester at the Norwegian University of Science and Technology. It was conducted in cooperation with Kongsberg Digital who provided me with the guidance, software, and hardware I needed to perform the experiments described in the text. My supervisor was Anastasios Lekkas, an associate professor of autonomous systems at the Department of Engineering Cybernetics. This work was done alongside three other courses and forms the foundation for my Master's Thesis that will be written in Spring 2020.

The following hardware was provided 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 dataset.

A workstation equipped with a NVIDIA 1080ti graphics card, provided by Kongsberg. It was used to train and experiment on the Mask RCNN instance segmentation model.

The following software resources were used:

Kongsberg Cogs.

3D models from Kongsberg K-Sim.

Matterport's Mask RCNN model implementation[1].

Ankurtaly's Integrated Gradients implementation[2].

Marcotcr's Lime implementation[3].

Scikit-image. For image processing in Python.

OpenCV-Python. For image and video processing in Python.

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 valuable guidance. From Kongsberg I want to thank Jostein Bø Fløystad for organizing the project, Christopher Dyken for making my time working with Cogs very pleasant, and Thorvald Grindstad for helping me with the 3D models. I look forward to working with you all next semester.

Table of Contents

Summary i

Sammendrag ii

Preface iii

Table of Contents vi

List of Figures ix

1 Introduction 1
- 1.1 Background and motivation . . . 2
- 1.2 Objectives . . . 3
- 1.3 Contributions . . . 3
- 1.4 Outline . . . 4

2 Theory 5
- 2.1 Computer Vision . . . 6
- 2.2 Machine Learning . . . 7
- 2.3 Deep Learning . . . 10
  - 2.3.1 Artificial Neural Networks . . . 10
  - 2.3.2 Convolutional Neural Networks . . . 11
  - 2.3.3 Mask RCNN . . . 13
  - 2.3.4 Feature Pyramid Network . . . 14
  - 2.3.5 ResNet . . . 15
  - 2.3.6 Training . . . 16
  - 2.3.7 Overfitting . . . 18
  - 2.3.8 Transfer Learning . . . 18
  - 2.3.9 Performance Measurements . . . 18
- 2.4 Synthetic Dataset Generation . . . 21
  - 2.4.1 Perlin Noise . . . 21

2.4.2 Normal Maps . . . 23
2.4.3 HSV Colorspace . . . 23

3 Explainable Arti cial Intelligence 25
3.1 Why Create Explainable AI Systems? . . . 27
3.2 What Is a Good Explanation? . . . 29
3.3 Interpretable Machine Learning . . . 30
3.4 Interpretable Instance Segmentation . . . 31
3.4.1 Jacobian Matrices . . . 32
3.4.2 Integrated Gradients . . . 33
3.4.3 Local Interpretable Model-Agnostic Explanations (LIME) . . . 35

4 Synthetic Training Data Acquisition 37
4.1 Kongberg Cogs . . . 38
4.1.1 Synthetic Data Generation in Practice . . . 38
4.1.2 The Architecture for Dataset Generation . . . 39
4.1.3 Terrain Generation . . . 40
4.1.4 Handling Small Boats . . . 41
4.2 Kongsberg K-Sim . . . 42
4.2.1 3D Models . . . 43
4.3 Cogs or K-Sim as a Synthetic Dataset Generator? . . . 43

5 Experiments 45
5.1 Experimental Setup . . . 46
5.2 Mask RCNN . . . 48
5.3 Robustness to Noise . . . 48
5.4 Feature Attributions . . . 49

6 Result and Discussion 53
6.1 Mask RCNN . . . 54
6.2 Robustness to Noise . . . 61
6.3 Feature Attributions . . . 62

7 Conclusion and Further Work 69

List of Figures

2.1 6

2.2 The MNIST dataset is often used to benchmark machine learning algorithms. It consists of 70000 image-number pairs. The gure is from Gradient-Based Learning Applied to Document Recognition[4]. 8

2.3 These datapoints have been clustered by the unsupervised learning algorithm k-means. The two different colors represent a different class. The gure was generated using a Python script and the Scipy package. 9

2.4 11

2.5 A convolutional neural network with two fully connected layers at the output for classi cation. The boxes represent feature maps, which are modi ed by kernel convolutions or pooling operations. 11

2.6 The pooling operations max pooling and average pooling. Here a 4x4 feature map is reduced to a 2x2 feature map. 12

2.7 The RCNN pipeline. The region proposal step, the feature vector generation, and the linear classi cation is shown. The Figure is from the RCNN paper[5]. 13

2.8 The Faster RCNN pipeline. The gure is from the Faster RCNN paper[6]. 14

2.9 The Mask RCNN architecture. The gure is from the Mask RCNN paper[7]. 15

2.10 The feature pyramid network. The gure is from Feature Pyramid Networks for Object Detection[8]. 15

2.11 The residual layer used in the ResNet architecture. The gure is from Deep Residual Learning for Image Recognition[9]. 16

2.12 Precision recall curve plots for different object detection algorithms. Taken from the VOC2007 paper[10]. 20

2.13 Figure illustrating the interpolation step in Perlin noise. It is meant as a supplement to the mathematics in this section. The gure is taken from Simplex Noise Demysti ed[11]. 21

2.14 The Hue Saturation Value (HSV) colorspace. The gure is from colorizer.org[12]. . . 24

3.1 Explanation in Arti cal Intelligence: Insights from the Social Sciences[13] describes Explainable Arti cial Intelligence (XAI) as in uenced by several research domains. This gure is from that paper and shows a venn diagram illustrating this concept. . . 26
3.2 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 classi ed as a wolf. The segments highlight the areas of the image which in uenced the classi cation the most. Figure is from "Why Should I Trust You? Explaining the Predictions of Any Classi er[14] 27
3.3 The ELO of the AlphaZero algorithm during training. Note that it exceeded the best previous algorithms. The gure is from Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm[15]. 28
3.4 Deep Inside Convolutional Networks: Visualising Image Classi cation Models and Saliency Maps[16] used the Jacobian matrix of an image classi er to generate the feature attribution shown above. 33
3.5 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 gure is from Axiomatic Attribution for Deep Networks[17]. 34
3.6 Shown above are three explanations generated for an image classi er 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 indicate Acoustic guitar, and to the right are segments that leads the model to classify the image as Labrador. The gure is from "Why Should I Trust You? Explaining the Predictions of Any Classi er[14]. 35
3.7 This gure shows the sampling of perturbed z's for a simpli ed two dimensional model. The gure is from "Why Should I Trust You? Explaining the Predictions of Any Classi er[14]. 36

4.1 38
4.2 The three parts of the dataset generation program communicates with Cogs through the Python-Cogs bridge. 39
4.3 The textures used when generating the image sample shown in Figure 4.1a and 4.1b. 41
4.4 A partially submerged sailboat. This issue caused biases early in the development of the system. 42
4.5 A screenshot of the K-sim application. The gure is from Henriksen's Master's thesis[18]. 43

5.1 The modi cation to the Mask RCNN network required in to use the XAI methods described in Chapter 3. 47
5.2 The input image and images corrupted by Gaussian noise. 49
5.3 The predictions to be examined in the feature attribution experiment. 50

6.1 The loss-function values experienced during training. Loss values are plotted against the y-axis, and the training epoch is increasing along the x-axis. The decreasing loss value indicates that the training process is working. 55

6.2 The loss-function values experienced during training. Loss values are plotted against the y-axis, and the training epoch is increasing along the x-axis. (On validation data) The decreasing loss value indicates that the training process is working. 56
6.3 Testing the performance of the instance segmentation algorithm. 57
6.4 The performance of the model on some real world images. 59
6.5 The confusion matrix generated by the network trained on the synthetic dataset. Generated with an Intersection Over Union (IOU) threshold equal to 0.5. 60
6.6 The input image for the Gaussian noise test and the resulting prediction. Note the false positive terrain detection. 61
6.7 The predictions obtained while gradually increasing the Gaussian noise in the input image. 63
6.8 The feature attributions for the containership prediction. 64
6.9 The feature attributions for the sky prediction. 65
6.10 The feature attributions for the sailboat prediction. 67

Abbreviations

AI Arti cial Intelligence. 2, 26, 27

ANN Arti cial Neural Network. 10, 11

AP Average Precision. 19, 20

API Application Programming Interface. 44

ASV Autonomous Surface Vessel. 2, 6

CNN Convolutional Neural Net. 11–14, 16, 18

COCO Common Objects in Context. 20

FC Fully Connected. 11–14, 16

FCN Fully Convolutional Network. 14

FPN Feature Pyramid Network. 15

GPU Graphical Processing Unit. 10

HSV Hue Saturation Value. vii, 24

IOU Intersection Over Union. i, ii, ix, 20, 58, 60, 61

LIDAR Light Detection and Ranging. 6

LIME Local Interpretable Model-agnostic Explanations. i, ii, 2, 4, 28, 31, 35, 48, 50, 62, 66, 70

mAP Mean Average Precision. 58

ML Machine Learning. 30, 31

RADAR Radio Detection and Ranging. 6

ReLu Recti ed Linear Unit. 10

RGB Red Green Blue. 12, 23, 24, 32

ROI Region Of Interest. 13, 14, 46, 62

RPN Region Proposal Network. 13, 14, 54

SLIC Simple Linear Iterative Clustering. 50

SVM Support Vector Machine. 13

VOC2012 Visual Object Classes Challenge 2012. 13

XAI Explainable Arti cial Intelligence. viii, 2–4, 25–31, 36, 46–48, 63, 66, 70

Chapter 1

Introduction

1.1 Background and motivation

As the world opens its eyes to the possibilities enabled by autonomous vehicles it has become clear that many challenges remain before this technology is ready. One of these challenges lies within the field of computer vision, the technology which enables computers to observe and react to their visual environment. This is not a new field, relatively speaking, but applying it to autonomous vehicles has revealed new requirements. A critical challenge is the explainability of these systems.

The visual complexity of real world environments has led to the development of complex computer vision systems to interpret them. Most state of the art computer vision systems utilize a method called deep learning. Deep learning is based on large networks of artificial neurons, which can contain upwards of several million artificial neurons and connections between them. These networks are trained on thousands of example images, which strengthens some connections and weakens others. The trained deep learning networks are able to detect complex visual patterns and produce accurate detections. However, these methods have one downside limiting their use in safety critical applications. Unlike explicitly programmed computer vision systems, for which the developer can guarantee the systems performance, deep learning based systems are so complex that the developer and end-user have to relate to one as a black box. This means that the system can only be described by its network architecture, the dataset used in the training process, and the training process itself. This limitation prevents the systems from being used in safety critical applications, which often requires explicit performance guarantees.

The maritime sector is one such safety critical application. A malfunction in navigation, control, or situational awareness can have dire consequences, both in terms of human safety and the environmental impact. In Norway the first implementations of large scale Autonomous Surface Vessels (ASVs) will likely be in coastal waters, an especially vulnerable area.[19] The legislation surrounding autonomous ships is still under development, but will likely contain several requirements regarding the explainability of the artificial intelligence systems used. This makes XAI an important enabling technology for autonomous ships.

Applying XAI to computer vision systems such as instance segmentation algorithms usually means that the system highlights which parts of the input image led to a certain prediction. This is often called feature attribution, and can be used to validate that a model has actually understood the general traits of some desired class, and is not just basing its predictions upon some undesirable bias in the training dataset. Several previous works have made progress on this, methods such as integrated gradients[17] and LIME[14] have both illustrated successful feature attribution, and will be examined in this thesis. In theory, enabling Artificial Intelligence (AI) systems to explain their inner workings opens many possibilities besides just system 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 new fundamentally new discoveries in many fields. In the far future, XAI might even become the connection between human beings and advanced general artificial 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.[20] Using such virtual environments to create datasets for computer vision algorithms has been explored in previous works, such as Henriksen's Master's thesis[18] 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 exibility with regards to ship selection, weather, background terrain, etc.

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. The work falls within the area of computer vision as an enabling technology for autonomous ships.

This can be simpli ed into four sub-objectives.

- Research the state of the art methods for computer vision and XAI systems.
- 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 instance segmentation models.
- Examine the trained models with XAI methods.

1.3 Contributions

This work implements a method for generating synthetic labelled training data using the Kongsberg Cogs graphics engine. It allows exible dataset production, both in terms of the image samples statistical properties and in terms of which classes/3D models are used. When running on a consumer-grade laptop with a NVIDIA GTX 1050ti graphics card it generates about 1000 images per hour. This new method is compared to the method developed in Henriksen's Master's thesis[18], which used Kongsberg K-Sim for syntethic dataset generation.

Using a synthetic dataset generated with the method introduced above, an instance segmentation algorithm called Mask RCNN is trained. The training process and prediction accuracy is then analysed. The Mask RCNN's performance on real world images is also tested, without any domain adaption having been performed. The trained Mask RCNN models robustness to Gaussian noise, which is often found in real world digital images, is also examined.

Some feature attribution methods, XAI methods which highlight which parts of an input contribute most to a prediction, are modified to work with the Mask RCNN instance segmentation method. These methods are: analysis of the Jacobian matrix, the integrated gradients method, and the LIME method. These are then used to analyse a handful predictions from the Mask RCNN and the resulting attributions are compared to determine the most interpretable method.

1.4 Outline

This thesis is split into seven chapters. Chapter one covers the background and motivation behind the work done in this thesis, highlighting the rapid development of autonomous systems and the need for XAI to make these newly developed methods interpretable to humans. Chapter two contains the theory necessary to understand the experiments performed, mainly focusing on computer vision, machine learning, and deep learning. Chapter three details the need for Explainable Artificial Intelligence, it explores what constitutes a good explanation, and how to create interpretable machine learning systems. Chapter four delves into the process of generating synthetic datasets using the Kongsberg Cogs graphics engine and compares Cogs and K-sim as synthetic dataset generators. Chapter five describes the experiments performed in this thesis, as well as the experimental setup with which they were performed. Chapter six presents and discusses the results of the aforementioned experiments. Chapter seven concludes the thesis and suggests improvements and ideas for further work.

Chapter 2

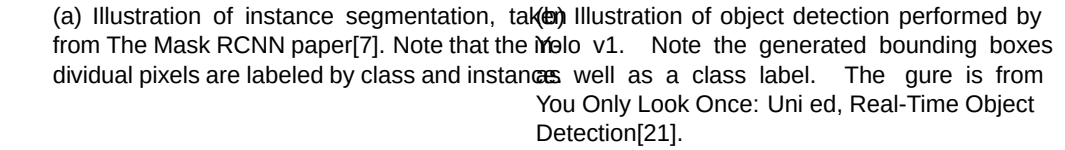
Theory

This chapter presents the relevant background theory needed to understand the thesis, the experiments, and the results. It begins by explaining the field of computer vision, the technology that allows computers to understand the visual world around them, and why it is critical 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. Machine learning based methods have benefited greatly from the increased amount of data existing in today's digital society, supplying the algorithms with plenty of training data to learn from. The chapter describes the different types of machine learning, with a focus on supervised machine learning, which is the method employed in this thesis.

The very popular field of deep learning is then covered. This is the technology which has enabled the huge performance increase seen within state of the art computer vision systems in the last decade. It is a method inspired by biological brains, in which the input data is propagated through several layers of artificial neurons to generate an output. Deep learning is a field that, thanks to the development of powerful processing units[1] and new methodologies, has seen rapid progression in the last years. The section highlights the deep learning methods used in this thesis, along with some of their defining features.

The final topic to be covered in Chapter 2 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.

[1] GPUs



(a) Illustration of instance segmentation, taken from The Mask RCNN paper[7]. 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 You Only Look Once: Unified, Real-Time Object Detection[21].

Figure 2.1

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, autonomous vehicles will also benefit from using computer vision. In modern systems cameras are one of several sensor systems used, it is 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. For ASVs this information is then used to form the situational awareness of the vessel[2]. This situational awareness contains deduced information about the movements and intentions of the 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 application, there are many tasks that computer vision systems 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 autonomous vehicles the following tasks are the most relevant.

[2] Along with other sensor data.

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 in an image as being part of a certain class.

Instance segmentation. Detecting each object in an image, predicting a class and a mask covering their silhouettes. An example is shown in Figure 2.1a.

Immediately one could assume that semantic segmentation would be the best fit for autonomous vehicles, since it classifies all the pixels of the image, thus extracting the most information from the input image. But this is not necessarily right, since it fails to separate different object instances. Semantic segmentation will not detect two cars if the cars are partially overlapping, instead it will predict a single group of car pixels. Instance segmentation will detect two different car instances (if it performs correctly), and its higher fidelity result is probably a better fit for autonomous vehicles.

2.2 Machine Learning

Creating self learning computers has been viewed as sort of a holy grail within computer science. The distinguishing feature between traditional algorithms and machine learning algorithms is that when programming machine learning algorithms the creator doesn't have to explicitly define the computing steps of the system, instead the computer will generate them independently based on some provided data. Of course the 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 main groups will be presented next.

Today machine learning is commonly grouped into three main types, distinguished by how they use the provided data to learn. These are:

Supervised learning

Unsupervised learning

Reinforcement learning

Since this thesis will focus on 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

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

Figure 2.2: The MNIST dataset is often used to benchmark machine learning algorithms. It consists of 70000 image-number pairs. The gure is from Gradient-Based Learning Applied to Document Recognition[4].

which, if the algorithm performs well is approximately equal to $h(x)$. This task is usually split into two types, classi cation and regression. In classi cation the input is to be assigned a class based on the input features. Normally the output is a probability distributions over the de ned possible classes. An example application is classifying hand-written digits. A dataset often used for this purpose is the MNIST dataset and is shown in Figure 2.2. The possible classes would in this case be the digits 0-9, and the input would be the images of the hand written digits. In 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, oor area, etc. The input would then be a vector describing these features, and the output value would be the price of the apartment.

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 framework of the machine learning algorithm[22]. 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 nal 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 over- tting. Over tting occurs when the algorithm memorizes the dataset instead of learning its general traits, this will degrade the algorithms 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

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

challenge of over tting will be described further in Section 2.3.7.

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 way the programmer doesn't have to provide desired answers. This reduces the work required to train the system. In supervised learning a substantial part of the work is collecting and labelling the datasets. For example: taking pictures of ships and labelling the ship types. 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 methods are often limited to clustering and pattern detection in the data. One such unsupervised machine learning algorithm is the k-means algorithm and is illustrated in Figure 2.3.

The nal type of machine learning 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 de nes a reward function which rewards the agent if it acts in a desired manner, and punishes it if it does not. The agent tries to receive as much reward as possible by optimizing its strategy[3]. After repeated attempts the agent has (hopefully)

[3]The strategy is often called policy.

created a successful strategy for maximizing the reward function.

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 combined with nonlinear activation functions. The thing that has changed lately, and made deep learning the dominant type of machine learning, is the accessibility 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. New network designs such as the ResNet architecture has also allowed higher performance deep learning algorithms. The ResNet architecture is described in Section 2.3.5.

2.3.1 Arti cial Neural Networks

Deep learning is based upon Arti cial Neural Networks (ANNs) which are networks of arti cial neurons, see Figure 2.4b for an illustration of an ANN. An arti cial neuron is a mathematical construct with several inputs and one output. It performs two simple operations. First it performs a weighted sum of the inputs before running the resulting sum through a non-linear activation function. In the weighted sum it is common to incorporate a bias value, which is independent of the input values. See Figure 2.4a for an illustration of such an arti cial neuron. The two operations performed by an arti cial neuron can be written as

$$z = \sum_{i=0}^{N} w_i x_i + b \tag{2.3}$$

and

$$y = \sigma(z) \tag{2.4}$$

where σ 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 facilitate easier explanation.

There are several different activation function in use today. Each with their own strengths and weaknesses. A few years ago the most popular one was the sigmoid function (2.5).

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.5}$$

However, the sigmoid function suffers from from an unfortunate 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 the training process of ANNs use these gradients, the training process can become very slow for large networks when using the sigmoid function. Because of this, the Recti ed Linear Unit (ReLu) (2.6) function has gained traction. It suffers less from the gradient saturation effect, even though its derivative is zero for all values of $z \leq 0$. The training process is described further in Section 2.3.6.

(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 2.4

Figure 2.5: 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.

$$(z) = \max(0; z) \tag{2.6}$$

For the final layer of a classification network it is common to use the softmax activation function. (Equation 2.7) It generates a probability distribution over all the layer outputs, ensuring $\sum_{i=0}^{N} z_i = 1$.

$$(z) = \frac{e^{z_i}}{\sum_{i=0}^{N} e^{z_i}} \tag{2.7}$$

Artificial neurons like the one 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. These layers are often referred to as Fully Connected (FC) layers. Sometimes these systems do not generate direct results, but rather feature vectors. A feature vector is a vector containing high semantic information extracted from the input which can be used further in a model.

2.3.2 Convolutional Neural Networks

For computer vision applications another deep learning architecture is common. These are Convolutional Neural Nets (CNNs), see Figure 2.5 for an illustration. CNNs differ from traditional artificial

Figure 2.6: The pooling operations max pooling and average pooling. Here a 4x4 feature map is reduced to a 2x2 feature map.

neural networks in that instead of using a weighted sum of all the previous neuron outputs they compute a weighted convolution of the input data. This allows them to apply spacial information (how detected features are positioned relatively to each other) in their calculations. Another advantage of CNNs is that they reuse the weights thanks to the repeated application of the same convolutions. This leads to a smaller memory footprint compared to equivalent traditional arti cial neural networks. If a CNN is used for classi cation it is common to use some FC layers at the end of the network. This is also illustrated in Figure 2.5. In this con guration the CNN is used as a feature extractor, while the FC layers perform the classi cation task.

There are two operations that de ne CNNs. One is the aforementioned weighted convolution and the other is the pooling operation. The weighted convolution is similar to the weighted sum in an arti cial neuron but work over an image patch instead of the inputs to a neuron. This can be written as

$$G[m;n] = ((f \quad h)[m;n]) = (\sum_{j} \sum_{k} h[j;k]f[m \quad j;n \quad k] + b) \tag{2.8}$$

where G is is the resulting matrix, f is the input matrix, $$ is a non-linear activation function, b is the bias, and h is the kernel[23]. Equation 2.8 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 (For instance the three Red Green Blue (RGB) layers in a color image). The kernel is a tensor (or a matrix in the case of a 2D input) that de nes the weight applied to each part of the image patch. Dependent on the values in the kernel, the kernel convolution can look for different patterns in the input matrix. Similarly to the FC architecture, CNNs also expose features of higher semantic value in the later layers. These matrices of high semantic value are usually called feature maps.

The nal operation that de nes CNNs is the pooling operation. This operation reduces the size of the input and thus the computational requirements for the entire system, while attempting to retain the semantic information. There are several pooling methods, differentiated by how they reduce the input size. The most common are max pooling and average pooling, these are showcased in Figure 2.6.

Figure 2.7: The RCNN pipeline. The region proposal step, the feature vector generation, and the linear classi- cation is shown. The Figure is from the RCNN paper[5].

2.3.3 Mask RCNN

In 2014 a research group from UC Berkeley released a new method for object detection[5]. The method combined a CNN classi er with a preprocessing step called region proposal. The region proposal step found the most ”interesting” parts of the image, ensuring that the ensuing classi er could work as effectively as possible. They called this method RCNN(Regions with CNN features). Note that RCNN architecture did not use FCs network for the nal classi cation, opting instead to use a Support Vector Machine (SVM), which is a type of linear classi er[5]. 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 Region Of Interests (ROIs) and were warped to t the CNN feature extractor, which demanded inputs of a certain size. The RCNN pipeline is shown in Figure 2.7.

The R-CNN 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 R-CNN and Faster R-CNN architectures which both improved on RCNN's inference speed and accuracy.

Fast RCNN, introduced in 2015, improved on the RCNN architecture. It replaced the SVM classi er with a FC network for determining the class and performing bounding box regression[24]. In contrast the original RCNN method simply used the bounding boxes from the region proposal step as object bounding boxes, by further re ning the bounding boxes the fast RCNN architecture achieved more accurate bounding box regression. 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 2.6. These improvements (plus some modi cations to the training process), improved inference-time, training-time, and accuracy.

In 2016, Faster RCNN further improved upon the Fast RCNN and RCNN architectures. It introduced a Region Proposal Network (RPN) to replace the region proposal step[6]. In the two earlier

Figure 2.8: The Faster RCNN pipeline. The gure is from the Faster RCNN paper[6].

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[25]. This is a reasonable fast method, but became the bottleneck in Fast RCNN. Faster RCNN solves this by training a separate model to propose ROIs. This RPN works on a feature map produced by the feature extractor. This further improves both execution time and performance.

Faster RCNNs modular architecture has facilitated easy modi cations. Mask RCNN is one such modi cation. It appends a new branch for predicting an object mask in parallel with the existing branches for bounding box regression and class prediction[7]. 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 modi cation 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 RoIPooling led to pixelwise misalignements in the ROI and the extracted feature maps[7].

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 does not use pooling layers or FC layers, opting to rather use specially con gured convolutional layers to reduce the size of the feature maps[26]. FCNs usually perform semantic segmentation, but since it works within a ROI it performs instance segmentation in Mask RCNN.

2.3.4 Feature Pyramid Network

The convolutional neural net approach has an inherit trade-off. As the input propagates through the network its semantic value becomes higher and higher, but the resolution is reduced due to the pool-

Figure 2.9: The Mask RCNN architecture. The figure is from the Mask RCNN paper[7].

Figure 2.10: The feature pyramid network. The figure is from Feature Pyramid Networks for Object Detection[8].

ing layers. In 2017 the Facebook AI Research group published a new network architecture which does not suffer from this effect[8]. This architecture is called a Feature Pyramid Network (FPN) and consists of two main components. The first component is a regular convolutional network, with increasing semantic value and decreasing resolution at the later layers. It then introduces lateral connections from the convolutional layers as well as upsampling from the later layers. This enables it to create feature maps with high semantic value and high resolution. Figure 2.10 shows this architecture. In their paper from 2017 FAIR demonstrated state of the art performance with this method. A feature pyramid network 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[1].

2.3.5 ResNet

The defining feature of Deep Learning is that the network gradually extracts information with higher and higher semantic value deeper into the network. This should allow the network to make decisions and calculations based upon high semantic information and thus achieve high performance. In

Figure 2.11: The residual layer used in the ResNet architecture. The figure is from Deep Residual Learning for Image Recognition[9].

theory, this should make deeper networks, that is networks with more layers of artificial neurons, better than shallow ones. There is only one problem. In practical use deeper networks do not perform better than shallow networks, often they even perform worse. When creating very deep neural networks the prediction accuracy first saturates, before it decreases rapidly as the number of layers increases further[9]. One might imagine that this reduced accuracy is caused by over fitting due to the increased number of parameters in the deeper network. However, experiments performed in the paper Deep Residual Learning for Image Recognition[9] found this to not be the case, since the performance on the training data also decreased.

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. They hypothesised that the reduced accuracy in deeper networks is caused by the network's inability to learn identity transformations[4]. This fact, combined with the difficulty in optimizing the parameters of the network layers, means that some network layers actually degrade the performance instead of improving it. To mitigate this, the researchers reformulated the network layers to learn residual functions with respect to the input instead of learning unreferenced functions[9]. This was done to make it easier for the network to learn identity transformations, which was theorized to improve performance. One such residual layer is shown in Figure 2.11.

2.3.6 Training

The behaviour of a machine learning system is decided by the values of its parameters. In an artificial neural network, for instance a CNN classifier, these parameters are the weights and biases in the kernels and the FC layers. When training a neural network these weights are gradually shifted

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

towards an optimal value. Note that this value might not be the global optimum value, but rather a local optimum. The example in this section will be based around a traditional fully connected artificial neural network.

The main method for training neural networks is called gradient descent. It minimizes a given loss function by calculating its gradient relative to the weights and biases and then nudging them in the opposite direction. This is shown in equation 2.9

$$w_{ij}^{n} = w_{ij}^{n} \quad \frac{@C}{@w_{ij}^{n}} \tag{2.9}$$

where w_{ij}^{n} is the weight between neuron i in layer n 1 and neuron j in layer n. is the learning rate, which decides how much the weights are nudged. $\frac{@C}{@w_{ij}^{n}}$ 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 2.9 only updates one weight, when a model is trained this function is usually applied to all weights and biases in the network several times.

There are several kinds of loss functions, each for different situations. Typically a deciding factor 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 specially designed to take advantage of this. If the network performs regression the output is typically a vector or a scalar, and again there exists loss functions designed for this purpose.

As seen in Equation 2.9 the change of a network parameter is proportional the to 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 function. 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 backpropagation.

When performing inference, that means using the model 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

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

if $z = f(y)$ and $y = g(x)$. Backpropagation uses this to find $\frac{@C}{@w_{i\ 1}}$ based on $\frac{@C}{@w_i}$. Where 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 is the backpropagation of gradients, is the main feature of backpropagation. This thesis will not go into more depth than this, for more information the reader is referred to Chapter 2 in Neural Networks and Deep Learning[27] for a comprehensive description.

2.3.7 Over tting

A problem plaguing not just deep learning models, but also machine learning models in general, is over tting. This occurs when the model memorizes the training dataset instead of learning its general features. This can be dif cult to detect since the performance of the model will continue to increase as normally when tested on the training dataset. A method for detecting over tting is to periodically evaluate the performance of the model on a separate dataset. The validation dataset is typically used for this.

A models tendency to over t the training data increases with a decreasing dataset size. One reason for this is that the model might pick up on patterns in present in a small dataset which is not present in the real world. This pattern might be attenuated if the training dataset is increased. Another factor is the complexity of the model. In general more complex models over t more frequently. 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 not exist in the real world.

Deep learning models are particularly susceptible to over tting. This is because of the extreme complexity of the models. Some con gurations of modern deep learning models, For instance ResNet, can have several million parameters[9]. A parameter can be a weight between neurons, a bias, or a weight in a kernel. This makes simply memorizing the dataset a valid strategy. Naturally, this strategy would not perform well in the real world and is therefore undesirable.

2.3.8 Transfer Learning

Normally a large amount of training data and time is required to train a deep learning model. This often restrict smaller experiments, like the one in this thesis. Fortunately there are methods that reduce the amount of training data and time required. This thesis will employ a commonly used method for this, called transfer learning. Transfer learning involves starting off with an already trained model and modifying parts of it to perform a desired different task. The practical details on how this is achieved is dependent on the network architecture being modi ed. As noted in 2.3.2, CNNs classi ers often consists of two parts, the feature extractor and the classi er. When performing transfer learning on a CNN classi er it is common to freeze the weights and biases in the feature extractor and only train the classi er network. This can drastically reduce the training data and time needed in the training process. The instance segmentation network used in this thesis is pretrained on the COCO dataset.

2.3.9 Performance Measurements

It is important to have a precise framework for comparing computer vision algorithms, to enable comparisons between different solutions. Mask RCNN performs three main functions, it classi es objects, creates a bounding box around each object, and creates a segmentation mask for covering each object. Because of model's multiple outputs several different performance measures are required to measure its full performance.

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

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

The performance measures for classi cation will be considered rst. A framework based upon true positives(tp), false positives(fp), true negatives(tn), and false negatives(fn) is often used for this purpose. This system is most easily illustrated with a table. In table 2.1 the relationships between ground truth and the classi ers output is shown. A well performing classi er would exhibit few false negatives and false positives. These measurements are often combined into precision and recall.

$$\text{precision} = \frac{tp}{tp + fp} \tag{2.10}$$

$$\text{recall} = \frac{tp}{tp + fn} \tag{2.11}$$

Precision indicates how accurate the classi er is by calculating the quotient of all correct positive classi cations and all positive classi cations. Recall, on the other hand, describes how well the model classi es all existing objects. This is done by calculating the quotient of all correct positive classi cations and all existing positive objects. This is shown in equation 2.11. A good classi er should have a precision and recall near one, that would mean that the classi er correctly classi es all existing objects in the input.

A common way to visualize these values are to plot precision vs. recall plots. The various precision and recall values are generated by tuning the sensitivity of the model, the method for this is speci c to each model. By examining a precision vs. recall plot one can gain an insight into how precise the model is at various sensitivity levels. The information of a precision vs. recall plot can be further condensed into an Average Precision (AP) value. It is de ned as

$$AP = \int_0^1 p(r)\,dr \tag{2.12}$$

where $p(r)$ is the precision at a recall value. This is equivalent to nding the area under the precision vs. recall plot. There are several ways to estimate this integral. The Pascal VOC2007 challenge used an interpolated precision value

$$p_{interp}(r) = \max_{\tilde{r}:\tilde{r} \geq r} p(\tilde{r}) \tag{2.13}$$

$$mAP = \frac{1}{11} \sum_{r \in \{0.0, 0.1, \ldots, 1.0\}} p_{interp}(r) \tag{2.14}$$

Figure 2.12: Precision recall curve plots for different object detection algorithms. Taken from the VOC2007 paper[10].

sampled over 11 recall values: 0.0, 0.1, 0.2, .., 1.0 (Equations 2.13 and 2.14)[10]. Figure 2.12 shows AP values calculated with this method, see the value in the parenthesis. Note that all the methods exhibit lower precision at higher recall values, this is because the increased sensitivity leads to more false positives.

When measuring the performance of classi er networks 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 done by plotting the ground truth classes along one axis of the plot and the actually predicted classes along the other axis. The actual predictions are normalized to represent a probability distribution. One way to use a confusion matrix is to rst 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 used. It measures how accurate a bounding box or a segmentation mask is relative to ground truth. IOU can be formulated as

$$IOU = \frac{BB \setminus GT}{BB \cup GT} \tag{2.15}$$

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 one, while a prediction that completly misses has an IOU equal 0. Depending on which benchmark is used, a prediction can be registered as correct or false. Usually an IOU above 50 is required. The Common Objects in Context (COCO) challenge, a popular benchmark for instance segmentation, uses ten different IOU

Figure 2.13: Figure illustrating the interpolation step in Perlin noise. It is meant as a supplement to the mathematics in this section. The gure is taken from Simplex Noise Demysti ed[11].

thresholds[28].

2.4 Synthetic Dataset Generation

When generating training data for use in deep learning, the core idea is to match the statistics of the generated data to the statistics in the real world. In other words, to make the simulated data as realistic as possible. Another important idea is to avoid any unintended biases in the data. For example: if an object detector is trained to detect oil tankers, but it is only trained on images of featuring red ships, the detector would learn to detect the color red and not the ship, this would make the detector 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 could be to create additional models by altering the color or texture of some existing model.

2.4.1 Perlin Noise

In procedural terrain generation, one of the main challenges is creating continuous, natural-looking formations. Most common noise generation methods suffer from the same problem. They create unrealistic, non-continuous terrain topologies. For example: a uniformly sampled heightmap might give 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. This problem highlights the difference between value based noise and gradient based noise. In order to create natural looking terrain the sampled height values need to in uence each other. In value based noise generators this is not the case, which leads to the unnatural looking jumps.

Gradient noise takes another approach. It creates a lower resolution noisemap of gradients, and recovers the high resolution gradient map by interpolation. This improves on value based noise since it forces the noise vary smoothly, which generates more natural terrain topologies. 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 closes grid points. However, it does not use a linear interpolation, instead using

$$f(t) = 6t^2 \quad 15t^4 + 10t^3 \tag{2.16}$$

as the blending function. See Figure 2.13 for a visualization of the interpolation step. The following mathematical formulation is from Simplex Noise Demysti ed[11].

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

$$i = \text{floor}(x)$$

$$j = \text{floor}(y)$$

this creates the distances

$$u = x \quad i$$

$$v = y \quad j$$

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

$$n_{00} = g_{00} \quad \begin{pmatrix} u \\ v \end{pmatrix}$$

$$n_{10} = g_{10} \quad \begin{pmatrix} u \\ v \quad 1 \end{pmatrix}$$

$$n_{01} = g_{01} \quad \begin{pmatrix} u \quad 1 \\ v \end{pmatrix}$$

$$n_{11} = g_{11} \quad \begin{matrix} u & 1 \\ v & 1 \end{matrix}$$

These values are then blended together using the blending function shown in (2.16)

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

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

to nally create

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

which is the nal value at the point P. 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.

2.4.2 Normal Maps

Normal maps are used to improve the detail of 3D models or heightmaps by encoding the surface normal vector at a point on the object, n_{XYZ} in the RGB channels of an image. This enables the lightning of small variations over the surface to be calculated, without increasing the number of polygons. In this project the normalmap was used to increase the detail level of the background terrain.

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

$$z = f(x; y) \tag{2.17}$$

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

$$F(x; y; z) = z \quad f(x; y) \tag{2.18}$$

which, the gradient (and the normal vector)

$$n = r\ F(x; y; z) = (\frac{@f}{@x}; \frac{@f}{@y}; 1) \tag{2.19}$$

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

$$G_x = \begin{bmatrix} 1 & 0 & 1 \\ 2 & 0 & 2 \\ 1 & 0 & 1 \end{bmatrix} \tag{2.20}$$

and

$$G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{2.21}$$

The gradient (2.19) is then L2 normalized

$$p(x; y)_{RGB} = \frac{n}{k n k} \tag{2.22}$$

and mapped to RGB values (0.0-1.0 for float and 0-255 for uint8).

2.4.3 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. This encoding, turned out to not be suitable for the requirements of the synthetic dataset generation. For example: to generate the color of some piece of background terrain one would have to sample three different

Figure 2.14: The HSV colorspace. The gure is from colorizer.org[12].

probability density functions, one for each RGB value. It is not easy to generate terrain with realistic colors when sampling RGB values. For example: how much blue should be present in the color of a rock? For humans this is not an intuitive question. This made the RGB probability distributions dif cult to de ne, and therefore the RGB encoding was not suitable for this application.

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 de ned on the interval 0 - 360, while the saturation and brightness values are often de ned 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 de ning the probability distributions for a piece of terrain one now de nes the limits of the hue, the saturation, and the brightness of the terrain. Each with a separate intuitive in uence on the resulting nal color. Note that HSV and HSV describes the same colorspace.

Chapter 3

Explainable Arti cial Intelligence

This chapter explores the eld of XAI and gives a quick introduction to its history as well as some of its 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. Various methods for constructing interpretable machine learning algorithms are discussed and how they can be adapted to use for instance segmentation.

Figure 3.1: Explanation in Arti cal Intelligence: Insights from the Social Sciences[13] describes XAI as in uenced by several research domains. This gure is from that paper and shows a venn diagram illustrating this concept.

The term XAI was rst coined in a paper released in 2004 by Michael van Lent.[29] It has since become the de facto term used when describing AI systems that attempt to explain their own behaviour. XAI is a new eld and unfortunately not many rigid de nitions exist. The problem however, has been around for some time. In the 1970s 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 distinct structure.

Along with the interest in expert systems, the interest in XAI also died down over the following decades. Then, in the 2010s, the deep learning revolution led to a large increase in complex machine learning models. This has led to a revival of XAI interest. Now that these complex machine learning methods are to be applied in safety critical operations XAI is poised to play an 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 de nitions. One de nition of interpretability is ”the ability to explain or to present in understandable terms to a human”.[30] This de nition de nes explainability as a part of interpretability. More speci cally 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 terms.

.

Figure 3.2: 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 classi ed as a wolf. The segments highlight the areas of the image which in uenced the classi cation the most. Figure is from ”Why Should I Trust You? Explaining the Predictions of Any Classi er[14]

XAI is a problem spanning several elds of research. Explanation in Arti cal Intelligence: Insights from the Social Sciences[13] attribute three contributing elds to XAI. Arti cial intelligence, Human-Computer Interaction, and Social Science. A Venn diagram describing this is shown in gure 3.1. This multi domain split contributes to the dif culty of assigning proper de nitions. Researchers from each of these elds have different point of views stemming from their different academic backgrounds. This leads to various de nitions and methods being used in research.

3.1 Why Create Explainable AI Systems?

If an arti cial intelligence 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 system operates as a black box without available explanations. This makes pin-pointing the cause of an error dif cult, if not impossible. This would make it dif cult to improve the 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 ensure this safety.

In the paper Why Should I Trust You?: Explaining the Predictions of Any Classi er[14] the authors show a good example of why XAI is valuable. In their experiment they trained an image classi er to separate images of huskies from images of wolfs. Their classi er performed well, it suc-

Figure 3.3: The ELO of the AlphaZero algorithm during training. Note that it exceeded the best previous algorithms. The gure is from Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm[15].

cessfully separated the images by subject. The researchers then used an XAI method called LIME to generate explanations for the classi cations. One of these explanations is shown in Figure 3.2. A quick glance at the image segments provided in the explanation allows even a non-expert to deduce that the classi er uses the snow on the ground to classify the image, not the features of the animal. This indicates that the model will not work well in the real world, in which images of wolfs 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 classi er example above, one can easily imagine that these XAI methods can be used to improve a model. By examining the explanations for the dog/wolf classi er the creator could quickly spot the bias in the model. This bias could then be removed by expanding the dataset to include images of wolfs 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 probably improve dramatically.

Having end users trust the AI system is important. In many professions classi ers like the ones discussed above would 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 this could have large consequences and the user would likely end up with a portion of the blame. If the classi er could provide explanations along with its predictions, the end user could more effectively evaluate their quality. This would not only improve the users trust in the system, but could also avoid potentially damaging errors.

The paper ”Peeking Inside the Black-Box: A Survey on XAI”[31] 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 knows algorithms in three different board games. (As seen in Figure 3.3) Maybe new though-processes and 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 is only a board game. But what if this algorithm was instead trained to treat dangerous illnesses? Then a good explanation could save potentially save millions of lives.

3.2 What Is a Good Explanation?

A natural next question is what constitutes a good explanation. Unlike many other questions within computer science this answer 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 create good explanations in the context of XAI systems.

Explanation in Arti cial Intelligence: Insights from the Social Sciences[13], 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 an 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 receivers 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.

The Simulation Heuristic[32] hypothesises that humans use internal mental simulation models in order to make 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 con rmation bias into account. By taking the con rmation bias into account one combats the human tendency to nd solutions that t 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 receivers mental model.

In the real world an event often has countless causes. It is the XAI system's role to lter through them all and only deliver the most important ones to the receiver, according to the guidelines de ned above. In computer vision the problem is somewhat simpli ed, since the possible causes are often limited distinct features of the input image. Still, generating intuitive explanations for classi ers is no easy task. Some methods for doing this will be presented in Section 3.4.

3.3 Interpretable Machine Learning

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

There are two main methods of creating interpretable machine learning systems. One method is to simply use models that are intrinsically interpretable. This can be achieved by using a modular design or by using a suf ciently 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 nal cat detection. This model could explain itself by simply highlighting the relevant parts of the cat that led to the classi - cation. But the problem remains, how would the system explain why it detected an ear at a speci c location? The modularity of the system would have to be quite ne grained to offer satisfactory explanations. An example of the alternate 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 a natural explanation 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, this might prevent it from detecting the complex patterns it otherwise would predict. For some problems this is not a problem, the performance loss is acceptable. But for more advanced machine learning tasks, such as instance segmentation in maritime environments, the performance loss is unacceptable and other methods must be used. This trade-off is called the intepretability-performance trade-off.

The second method for achieving interpretability are ad-hoc methods. These methods do not modify the existing ML system. Instead they systematically modify the input data while observing the outputs to generate explanations for the model. These methods have several advantages over the intrisic methods. For one they are not subject to the interpretability-performance trade-off since they do not modify the model being examined. Second, they are often model-agnostic. This is a term used when an XAI method can be used on all ML models. This exibility is desirable. A downside is that these methods are inherently less ”elegant” than the intrinsic methods, since they add extra programming 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 the 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 parts of the model in uence decisions. Which features of the system leads to which predictions. In other words: how the model works. These methods are usually model speci c since they use the speci c features of the model in their explanations. Prediction explanations aim to inform why a model reached a speci c decision. For instance: a sail-boat 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 rst model. This might seem counter-intuitive at rst: 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. LIME does this by training a ML model to explain one of the primary models predictions. It then uses the parameters of the interpretable secondary model to infer which parts of the input strengthens the classi cation and which inputs weaken it[14]. 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 can't 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 the local and the global surrogate model methods are model-agnostic XAI methods.

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. The method integrated gradients employs a similar method for highlighting the most important parts of an image[17]. Integrated gradients is one of the methods that will be further discussed in Section 3.4.

3.4 Interpretable Instance Segmentation

The Mask R-CNN instance segmentation method described in Section 2.3.3 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 classi er network in the explanations. The general principles of each method still applies. Image classi cation is a simpli cation of the object detection algorithm illustrated in Figure 2.1a. It does not create bounding boxes around the objects and is limited to one detection per image. The details of implementing the methods for the instance segmentation application will be covered in Section 5.1.

Humans are visual creatures. Therefore, a natural way for a computer vision algorithm to explain itself is to highlight the parts of the input image that most in uenced the result. There are other ways to gain insight into the model, an alternative method could be to nd 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 de ned by a handful of visual features, such as the sky, or the ocean. However, that method is outside the scope of this thesis and is instead recommended as a possible topic for further works.

Feature attribution is a eld of research which 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 de ning feature of the sailboat class. A subclass of feature attribution are saliency maps, which highlight the areas that most in uence the prediction on

a pixel by pixel level. Two of the methods that will be evaluated in this thesis produce such saliency maps.

3.4.1 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 classi er can be seen as a function

$$f : R^n \; ! \quad R^m$$

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.

$$J = \begin{matrix} \frac{@f}{@x_1} & \frac{@f}{@x_n} \end{matrix} = \begin{bmatrix} \frac{@f_1}{@x_1} & \frac{@f_1}{@x_n} \\ \vdots & \ddots & \vdots \\ \frac{@f_m}{@x_1} & \frac{@f_m}{@x_n} \end{bmatrix}$$

If a classi er network takes as input an RGB image with 10^6 pixels and classi es the images with regards to two possible classes. The classi cation function will be equivalent to

$$f : R^{3 \quad 10^6} \; ! \quad R^2$$

if the output values represent the probability of the input being part of a given class. The Jacobian matrix can then be estimated numerically by the approximation

$$\frac{@f}{@x_i} = \lim_{h \; ! \; 0} \frac{f(x_{i0} + h) \quad f(x_{i0} \quad h)}{2h}$$

where x_{i0} is the x value where the derivative is to be calculated and h is a suf ciently small constant.[1] 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 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.

This method does however, comes with some drawbacks. One being that it is a linear estimation of the function, and it is only valid for small perturbations since it does not predict non-linear effects. This is unfortunate, since the classi er is a highly non-linear mapping 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 component might cover several pixels and require large pixel-wise changes in order to disappear into the background texture of the image. The Jacobian matrix will not not necessarily describe the effect of these changes accurately. This limits the Jacobian matrix's ability to highlight the important

[1] Many deep learning software programs also has gradient calculations build in.

Figure 3.4: Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps[16] used the Jacobian matrix of an image classifier to generate the feature attribution shown above.

object components present in an image. Still, as shown in figure 3.4, the attribution offered by it can still offer insight into the model.

3.4.2 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 x^0 in addition to the input image x. The method then gradually steps the input from the baseline value x^0 towards the actual input 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 similar situation with the relevant feature absent[17].

Integrated gradients consider a straightline path from x^0 to x and compute 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[17]. 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 (pixel) is defined as

$$\text{IntegratedGrads}_i(x) := (x_i - x_i^0) \int_{\alpha=0}^{1} \frac{F(x^0 + \alpha(x - x^0))}{x_i} d\alpha$$

where α is an interpolation variable and $\frac{F}{x_i}$ is the derivative of the output with respect to a dimension (a pixel value)[17]. This integral can be approximated by a series of sums

Figure 3.5: 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 gure is from Axiomatic Attribution for Deep Networks[17].

$$\text{IntegratedGrads}_i(x) \quad (x_i \quad x_i^0) \quad \sum_{k=1}^{m} \frac{F(x^0 + \frac{k}{m} \quad (x \quad x^0)) \quad \frac{1}{m}}{x_i}$$

in which m represents the number of steps used in the approximation[17]. Figure 3.5 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 were enough to provide a good result[17]. This means that the integrated gradient method is at least 20-300 times slower than examining the Jacobian method described in the previous section. Section 6.3 examines whether this extra computing cost is worth it.

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 affect the output. To achieve this, the algorithm must work on super-pixels instead of individual pixels and analyse how changing batches of pixels affect the output. The method described in the next section does just this.

Figure 3.6: 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 "Why Should I Trust You? Explaining the Predictions of Any Classifier[14].

3.4.3 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. [14] 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 3.6 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 $L(f; g; \pi)$ 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' is defined as the presence or absence of the image segments. Imagine that the image is segmented into N segments, then $x' \in \{0,1\}^N$, representing if 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 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 3.7, 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 3.7: This figure shows the sampling of perturbed z^0s for a simplified two dimensional model. The figure is from "Why Should I Trust You? Explaining the Predictions of Any Classifier[14].

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

The LIME algorithm doesn't explicitly specify what kind of surrogate model is used. In their paper however, a linear classifier is used. This defines a weight vector $w \in R^m$ where m is the number of features selected to train the surrogate model. The classification is then defined as

$$g(z^0) = \begin{cases} 1 & \text{if } w \quad z^0 > T; \\ 0 & \text{otherwise} \end{cases}$$

where T is some threshold value. When training, the values of 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 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 used is left to the users discretion. One downside mentioned in the LIME paper[14] is that sometimes image segments can't 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.

Chapter 4

Synthetic Training Data Acquisition

This chapter presents the program that generates synthetic datasets using the Kongsberg Cogs graphics engine. It also discusses the parameters that can be varied between image samples, the procedural terrain generation, and how small objects are handled in the program. Towards the end of the chapter, Kongsberg K-Sim is discussed and compared to Kongsberg Cogs as a tool for synthetic dataset generation.

(a) A scene generated by the Cogs 3D rendering engine.

(b) The accompanying object masks.

Figure 4.1

4.1 Kongberg Cogs

Cogs is an in-house 3D rendering-engine developed by Kongsberg Digital. It is designed to be a lightweight and exible renderer for use in their projects. It provides vertical integration, meaning that the team at Kongsberg Digital has full control of every part of the engine and can guarantee its operation. It is designed to be exible and modular, making it easy to modify for special projects, such as generating synthetic training data for machine learning applications.

4.1.1 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 con guration via a Python API. This API supports asynchronous communication between a Python script and the Cogs engine. During the thesis this bridge has been expanded as needed, thanks to continued support from contacts at Kongsberg Digital.

The Cogs engine enables dynamic loading and removal of 3D models. These can then be placed and rotated as desired. Cogs also includes an ocean simulation, in which the wave height and period can be changed. 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 water color

the fog strength

the fog color

the terrain shape

the terrain color

the sun location

the camera rotation

the camera height

the camera eld of view

the appearance of the sky

Each of these can be sampled from a chosen probability distribution, during testing multiple combinations of parameters were tested. These parameters can be accessed with Python code, which allows the parameters to be changed during the dataset generation process.

4.1.2 The Architecture for Dataset Generation

The Python program that interfaces with Cogs to produce a synthetic dataset consists of three main parts. These parts are the Scene-Populator, the Scene-Randomizer, and the Sample-Generator. The architecture is shown in gure 4.2. They each communicate with Cogs through a separate Python bridge as described in the previous section.

Figure 4.2: 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 would 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 places the models in a random location in front of the camera. It ensures that they do not overlap, and samples all the parameters in the simulation according to their respective probability distributions. It also generates the background terrain, with the method described in Section 4.1.3. It also ensures that small objects are visible, this process is described in Section 4.1.4.

After a scene has been prepared by the Scene-Populator and the Scene-Randomizer, the Sample-Generator downloads the scene image and the scene segmentation via the Python bridge as .png files. The Sample-Generator is also responsible for keeping track of the complete dataset and logs how many times the different classes appear. Each generated sample consists of three files: the scene image, the scene segmentation image, and a .json file containing a mapping from pixel value to class for the scene segmentation image. After the three files have been saven the program resets and the next sample is generated.

4.1.3 Terrain Generation

Cogs has a built in terrain renderer. It requires three textures to operate: a heightmap, a colormap, and a normalmap. The heightmap describes the height of the terrain at a given point in the scene, the colormap describes the color at a given point, and the normalmap increases the graphical fidelity of the terrain by encoding the surface normal as an RGB image.

The heightmap 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 would yield discontinuous, unnatural looking terrain. The initial Perlin noise is then filtered to further smoothen 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. Perlin noise is further described in Section 2.4.1.

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 noise to induce variations within the color. The green terrain types are sampled in the HSB color space, which makes it easier to sample natural looking hues. During the image sample generation

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

(b) The heightmap 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 normalmap generated by the terrain generation.

Figure 4.3: The textures used when generating the image sample shown in Figure 4.1a and 4.1b.

the terrain type is selected through sampling a con gurable probability distribution.

4.1.4 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 the same 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 model 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 signi cant wave height is 12 meters. Naturally the discussion above would be different when simulating a small eld of view camera (a camera with high zoom), but it is meant as a guiding example. This means that the smaller vessels should be placed closer to the camera 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 oat on top of the water, but in Cogs objects have xed 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 anyways. But for small boats this could cause the entire hull to be submerged under water. This led to a bias in the dataset which became especially visible for the sailboat class. Because of their small size the model had never seen a non- ooded sailboat and when it encountered real world images of sailboats with their hulls visible it classi ed them incorrectly.

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

Figure 4.4: A partially submerged sailboat. This issue caused biases early in the development of the system.

To alleviate both problems described in this section, the dataset generator was modi ed to measure the size of the 3D models it placed. If the measured size of the object was below a certain threshold it would reduce the object's distance to the camera with a set probability. If it did move the object closer it would also lift the 3D model higher in the water by translating it in the Z-direction. These parameters can be con gured by the user along all the other parameters described in this chapter.

4.2 Kongsberg K-Sim

Kongsberg K-sim is a commercially available training simulator for maritime personnel. It is the software used in Henriksen's Master's thesis[18] to generate synthetic training data. Since its main purpose is to be a maritime simulator it has features which are not immediately bene cial for synthetic data generation. These features include: a state of the art hydrodynamics system, support for towing and tugging, a ship bridge simulator, support for hardware simulation, and much more. It also lacks some critical features which made its use as a synthetic dataset generator challenging. The process of generating synthetic datasets with K-Sim is comprehensively described in Henriksen's Master's thesis[18]. A comparison between Cogs and K-sim as synthetic dataset generators is presented in Section 4.3.

Figure 4.5: A screenshot of the K-sim application. The gure is from Henriksen's Master's thesis[18].

4.2.1 3D Models

Since K-sim is a maritime simulator it needs high quality 3D models. Many of these models mirror real world vessels, which further improves the realism. About 110 models from K-sim has been made available for use in this thesis. The models represent several different ship types, as well as offshore wind turbines. A minor annoyance was that the K-sim 3D models are in the OpenFlight (. t) format, a format incompatible with Cogs. This required them to be converted manually to the Wavefront (.obj) format, which Cogs is compatible with. This was a time demanding process. When the models had been converted, the visual delity of the generated datasets increased dramatically, making the manual labor well worth the effort.

4.3 Cogs or K-Sim as a Synthetic Dataset Generator?

In this section Cogs and K-sim will be compared as tools for synthetic dataset generation. The evaluation of the Cogs system is based on the experiences during this project and the K-Sim information is from Chapter 4 in Henriksen's Master's thesis[18], in which K-Sim was used for the same purpose.

Cogs and K-sim are very different software tools. While Cogs is an in-house lightweight 3D graphics engine, K-sim is a feature-rich commercially available maritime simulator. Immediately Cogs seem like the easier tool to use for synthetic dataset generation, but it has some downsides. These include:

the lack of buoyancy simulation. this caused the bias described in Section 4.1.4. This is not a problem in K-sim, as it incorporates advanced buoyancy simulation.

no bow wave simulation. It is unclear how much of an effect this would have on predictions. It might theoretically impact velocity estimations, if such were to be implemented in the future. In K-sim bow waves are simulated, but it is unclear how advanced this simulation is.

no velocity simulation. This makes it more dif cult to create test videos, since the velocity and movement over time must be handled outside Cogs. Due to Cogs' modular design it would probably be possible to modify it with a physics engine to enable this. Since K-sim is a physics simulator it includes velocity simulation.

no built in harbour environments. These would have to be created outside Cogs and loaded as objects alongside the vessel models. K-sim contains several high delity models of real world locations such as harbours and coastal areas.

These downsides are minor and thanks to Cogs modular design many of them can be alliviated with a later modi cation. When inspecting K-sim it quickly becomes clear that it was not designed for synthetic dataset generation, as it lacks some important features:

lack of built in screen recording or screen captures. Neither scene images or segmented images. This led to the development of a convoluted process for extracting both scene images and segmentation masks from the simulation. The author of Henriksen's Master's thesis[18] noted that this process was challenging task and involved quite a bit of manual oversight.

no built in Application Programming Interface (API). This meant that the simulation parameters had to be changed manually during the dataset generation period.

no mask generation for sky, water, or terrain. It is uncertain whether this could be implemented for K-sim as it was not attempted in Henriksen's Master's thesis[18].

For a full description of the K-sim dataset generation process see Henriksen's Master's thesis[18] Chapter 4. It details the large amount of work required before K-sim could be used to generate synthetic datasets. In comparison Cogs has been relatively easy to work with.

Chapter 5

Experiments

The following chapter presents the experiments performed in this work, as well as the experimental setup with which they were performed. The experiments aimed to explore:

- how well the instance segmentation algorithm Mask RCNN performs when trained on a synthetic dataset.
- the trained Mask RCNN model's robustness to Gaussian noise.
- how well the Jacobian matrix works for feature attribution.
- how well the integrated gradient method works for feature attribution.
- how well the LIME feature attribution works.

These experiments were performed during the development of the systems and are not meant as exhaustive studies into their respective topics. Nevertheless, they do offer some interesting insights into the behaviour of the systems.

5.1 Experimental Setup

The instance segmentation algorithm for the experiments in this chapter was trained on 23500 synthetic images, which was divided into two groups. One consisted of 22000 images and was used for training while the remaining 1500 was used for validation and testing. These images were generated using the procedure described in Chapter 4 and used 150 different 3D models of ships, boats, and offshore wind turbines. The classes represented in this dataset and the number of 3D models describing each class are shown in Table 5.1. The models were sampled according to the method described in Section 4.1.2 in order to avoid class imbalances.

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

The generated images had a resolution of 1024 by 1024 pixels and the final dataset occupied 51.7 GB of disk space. The training of the instance segmentation and XAI evaluation was performed on a workstation provided by Kongsberg Digital. It was equipped with an NVIDIA 1080ti graphics card, as well as an Intel i7-7700k CPU and 32 GB 2133 MHz RAM. This workstation was shared with another student.

As a reminder, the Mask RCNN algorithm 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 2.3.5. The second part of the algorithm 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 regions of interest. The third and final layer then examines the parts of the feature map noted as a ROIs and predicts the class, bounding box, and the mask for each. This means that in a sense the Mask RCNN algorithm can be viewed as several simple classifiers, one for each ROI.

Since the XAI methods used in the experiments described in Section 5.4 were originally designed for simple one object classifier networks and the Mask RCNN algorithm is a complex model consisting of several sub-networks working together, some simplifications and assumptions had to be made in the experiments performed for this thesis. Internally, the Region Proposal network evaluates a set number of fixed anchors in the input feature map and computes the corresponding bounding box and predicts whether it contains a foreground object or a background object. This process of passing regions of interests on to the final class prediction network is the defining feature of RCNN networks and is explained further in Section 2.3.3. The XAI experiments performed in this thesis focuses on the class prediction for a given prediction. Therefore, the output from the Region Proposal network has been frozen in the experiments. The 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 region proposal network. Figure 5.1a shows the original Mask RCNN architecture and Figure 5.1b shows the modified architecture performed in this thesis.

(a) Original Mask RCNN

(b) Modi ed Mask RCNN

Figure 5.1: The modi cation to the Mask RCNN network required in to use the XAI methods described in Chapter 3.

When using integrated gradients and LIME, which both alter the input image in some way while either calculating multiple gradients or classi cations scores, an assumption is required. In normal operations the nal region of interest 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 class prediction network, and might lead to a different prediction result. The assumption that is made when using integrated gradients and LIME in this way is that this variability does not impact the XAI method's ability to generate explanations for the model predictions.

5.2 Mask RCNN

The instance segmentation algorithm Mask RCNN was trained on the data described in the previous section for 10 hours and 36 minutes using the GPU for hardware acceleration. This thesis uses the Matterport Mask RCNN implementation[1] which is available open source on Github. After the system had trained on the 22000 synthetic training images it was tested on the synthetic validation dataset as well as real world images and videos. This was done without any domain adaptation which would likely improve the performance further as found in Henriksen's Master's thesis[18]. The model was con gured to use the ResNet101 feature extractor and to expect square images of 1024 by 1024 pixels. It was trained for 400 epochs with 100 training steps per epoch. The learning rates used are shown in Table 5.2. A learning momentum of 0.9 was used throughout the training process. The various loss function values were logged using Tensorboard and will be presented as a part of the result. The initial model weights were pretrained on the COCO dataset.

Epochs	Learning rate
0-100	0.001
100-200	0.0005
200-300	0.00025
300-400	0.00003125

Table 5.2: The learning rates used while training the Mask RCNN network.

5.3 Robustness to Noise

An easy way of testing the performance of the system under non-ideal circumstances is to design an experiment in which the input images are gradually more and more corrupted by a Gaussian noise. During this process the classi cation scores are logged and used as an indicator for the models robustness. This experiment was repeated 10 times and the mean prediction scores at each noise level was calculated and used in a graph for visualization. Note that this experiment was performed with the restrictions on the region of interest network described in Section 5.1, with the simpli cations that entails.

Since the model has only been trained on synthetic images it has never observed Gaussian noise in its training data. Gaussian noise is extremely common in real world images since it a byproduct of digital camera sensors. The experiment also sheds light on whether the model has learned robust general features describing the classes or is basing its predictions on weaker features. If so, predictions for certain classes are expected to decrease in quality more rapidly than others.

(a) (b) (c) (d)

Figure 5.2: The input image and images corrupted by Gaussian noise.

Figure 5.2a, 5.2b, 5.2c, and 5.2d, shows some of the images used in the experiment. The leftmost image is the original image, the center-left one is corrupted with Gaussian noise $\mathcal{N}(0, \sigma = 50)$, the center-right one is corrupted with Gaussian noise $\mathcal{N}(0, \sigma = 100)$, and the one to the right is corrupted with Gaussian noise $\mathcal{N}(0, \sigma = 150)$. The quality of the prediction is expected to decline as the noise increases, but how it declines is of interest. One unnatural way for the system to fail would be if the system got stuck with some false class. If it became more and more certain that the detected objects were of some incorrect class as the image quality decreased. This could indicate a bias in the dataset, and would be unfortunate since the system would not be aware of its own failure. Normally a human observer would know if it was uncertain about prediction and allow precautionary measures to be taken. An instance segmentation algorithm should behave the same way, otherwise a system which uses the model could not trust any of the models prediction since they they could be completely wrong even if the model predicts a class with 100% certainty.

A more desirable way for the system to fail would be if it gradually lost faith in its original prediction, and entered a period of uncertainty before not detecting an object at all. This indicates that the model is aware of its own limitations and thus a system which uses the model's predictions could take this inaccuracy into account. As it turns out, the system does just this, go to Chapter 6 further discussion around this.

5.4 Feature Attributions

The last experiment performed in this thesis is a comparison between several feature attribution methods to determine if they are able to increase the interpretability of the Mask RCNN instance segmentation predictions. As a reminder: feature attribution means highlighting the parts of the input image that most in uence a prediction result. This should give the user an idea of whether the model has understood the underlying characteristics of an object class and is looking at the expected areas of the image to reach its conclusion. In other words: whether the model has correctly learned the general features of the objects in the dataset. If it has, it should generalize well. To test the methods they were used to generate feature attributions for three different predictions. These are

(a) The sailboat prediction is examined.

(b) The sky prediction is examined.

(c) The containership prediction is examined.

Figure 5.3: The predictions to be examined in the feature attribution experiment.

shown in Figure 5.3.

The Jacobian matrix represents the simplest type of feature attribution to be examined in this thesis. As noted in Section 3.4.1, it is simply the output classi cation derivative with respect to the input pixels of the image. Intuitively, the pixels that lead to the largest change in output classi cation score are the most important. The integrated gradients method is a simple extension of the Jacobian matrix method. It de nes an integral of the gradients from some baseline input to the actual input image and generates a feature attribution that encodes more information than the simple Jacobian matrix. See Section 3.4.2 for more information. The integrated gradients method was con gured to use 50 steps when estimating the path integral. The gradients were calculated using Tensor ow's built-in gradient function.

As discussed in Section 3.4.3, LIME takes a different approach to interpreting object classi ers, opting to use an interpretable surrogate model trained on the input image segments instead of the input/output gradient. According to the LIME paper[14] this should provide interpretable explanations. LIME was the third and last feature attribution method tested in this experiment.

LIME is a exible method that can be customized to t the needs of a user. For this experiment it was con gured to use the Simple Linear Iterative Clustering (SLIC) image segmentation algorithm. SLIC was chosen because of its ease of use, ef ciency, and tendency to create segmentation patches of uniform size. It works by encoding the pixel values in a 5D space de ned by the CIELAB colorspace and the x,y position in the image. It then performs a clustering operation that groups pixels based on a distance measure[33]. For this experiment, SLIC was con gured to split the image into 250 segments. LIME was con gured to highlight the 5 segments which in uenced the prediction the most. Based on the original input image, 1000 altered image-prediction pairs were computed and used to train the LIME surrogate model.

As seen in Section 3.4.1 and 3.4.2, both the Jacobian matrix and result from the integrated gradients method are matrices. To maximize their interpretability they must be processed before visual-

ization. This can be done in various ways, but for this experiment the following simple process was used.

First the values of the elements in the matrix are converted to absolute values.

Then, for each pixel, the average value of the RGB channels is found.

The result is then normalized to between 0 and 1.

Finally a threshold is applied. If a value is below the threshold value, it is set to zero, otherwise it remains the same. In the experiment this threshold value was set to 0.1.

Chapter 6

Result and Discussion

6.1 Mask RCNN

As described in Section 5.2 the Mask RCNN was trained for 400 epochs with a incrementally decreasing learning rate. (shown in Table 5.2) Figure 6.1 shows the loss function values experienced during the training process. Note that the less opaque plots represent the actual recorded loss values, while the more de ned ones are smoothed. Figure 6.2 show the loss functions when the model is tested on the validation data. The increased noise is because a smaller sample size was used when calculating the loss values on the validation data. (100 vs. 5) The model was trained on an aggregate loss function consisting of loss functions describing the class, bounding box, mask, RPN class, and RPN bounding box prediction errors.

From Figure 6.2a it is clear that the model improves most during the rst 100 epochs. The decrease in learning later in the training process is probably due to the model saturating, as well as the gradual reduction in training rate. In the future the performance might be increased by expanding the dataset (an easy task because of its synthetic origin), using dataset augmentation, ne tuning the training process, or improving the model architecture. Figure 6.2e shows that the loss values for the RPN class predictor, the part of the network that decides whether an object is present at a location in the feature map, appears very noisy. It is speculated that since the loss values are so low, combined with the small batch size, individual detection errors might impact the plot heavily.

Figure 6.3 shows the performance of the Mask RCNN algorithm after the training regime described previously described. The model exhibits good performance. The examples in Figure 6.3 have been chosen to highlights some interesting details. In the rst test, the model was presented with a challenging scene to analyse. It pictures a containership partly occluded by an icebreaker. To achieve good performance the model would have to understand that there are only two ships present and combine the two visible parts of the containership into one complete ship. The model got halfway there, it successfully separated the icebreaker from the containership, but did not combine the two visible containership segments into one ship. Instead it detected two containerships, observe that the segments representing the detection has different colors. During the testing process the model was subjected to many such examples, sometimes it performed correctly and sometimes it got it wrong. During testing the model often performed better when the occluding object was small, or when the two visible segments of the occluded object were connected. All the class predictions in the image were correct, as were the predicted masks, if the previously discussed error is ignored.

In the second test, the image contains a large passenger ship and a sail boat. The sail boat is far away and appears very small in the image. Still, the image is meant to represent a typical 'easy' input image. The model performed well, by detecting all the objects in the image and predicting the correct class as well as accurate instance masks. This is typical, the trained model handles simple scenes like this one very well. When observing the ocean prediction it is clear that the predicted mask does not reach all the way to the edges of the predicted bounding box, which is represented by the dotted line. This issue remains for all predictions near the edges of the image.

In the third and nal test on synthetic images, the model was presented with an image containing many small boats as well as some terrain in the background. It correctly classi ed all the small boats (the small white one is actually a small ferry), but detected two separate instances of terrain. This double detection error happen occasionally with the terrain class. It could probably be im-

(a) Sum of all training losses during training.

(b) Class predictor loss during training.

(c) Bounding box predictor loss during training.

(d) Mask predictor loss during training.

(e) RPN class loss during training.

(f) RPN bounding box loss during training.

Figure 6.1: The loss-function values experienced during training. Loss values are plotted against the y-axis, and the training epoch is increasing along the x-axis. The decreasing loss value indicates that the training process is working.

(a) Sum of all training losses during training. (On validation data)

(b) Class predictor loss during training. (On validation data)

(c) Bounding box predictor loss during training. (On validation data)

(d) Mask predictor loss during training. (On validation data)

(e) RPN class loss during training. (On validation data)

(f) RPN bounding box loss during training. (On validation data)

Figure 6.2: The loss-function values experienced during training. Loss values are plotted against the y-axis, and the training epoch is increasing along the x-axis. (On validation data) The decreasing loss value indicates that the training process is working.

(a) Test input from synthetic dataset

(b) Detections

(c) Test input from synthetic dataset

(d) Detections

(e) Test input from synthetic dataset

(f) Detections

Figure 6.3: Testing the performance of the instance segmentation algorithm.

proved upon by non maximum suppression, or by implemented a more advanced terrain generation with varied textures. Also note that sometimes the predicted masks of separate detections overlap. This often occurs with small boats like the ones visible in this example. This is because there is no communication between the different final mask predictions. This issue could be reduced somewhat with a filter that always places object detections on top of the ocean detection. This would not reduce accuracy since the boats typically float on top of the ocean anyways. But the fundamental issue would still remain, since it is inherit to the Mask RCNN architecture. Performing semantic segmentation or panoptic segmentation[1] instead would also remove this problem, since for those methods each pixel can only be assigned one class. However, as mentioned in Section 2.1 semantic segmentation does not separate different object instances, this might make it unsuitable for many applications.

In addition to the tests on the synthetic dataset, the model was also exposed to some real world images. It is important to note that the model has never been been trained on real world images during the training process (Except the initial weights, which were pretrained on the COCO dataset) and that no domain adaption has been performed. The results are shown in Figure 6.4. The first image contains a clearly visible sailboat with two people on board. In the background there is terrain. Note that the sky is not visible at all. For this image the model performed terribly, it mistook the background terrain for sky, it classified the sail boat as a tanker, and the two people on board were classified as tug boats. The red car on the beach is classified as a motorboat, this might actually indicate a bias in the motorboat class because of the red color, and a portion of the beach is labelled as an oiltanker. The good news are that the model at least detects objects in the image, it is mainly the classification that is incorrect. This image is very different to the ones it had trained on, and as expected the model performs poorly. This example represents a very bad prediction result.

The model performs better on the second image. The image contains a clearly visible cruise ship out at sea. The model correctly classified the ship as a passenger ship, but unfortunately detected a part of the ship as another ship. The predicted ocean mask is also not entirely accurate. This image represents an image that is more similar to the one in the training data, and the model was expected to perform better on it.

As described in Section 2.3.9, the Mean Average Precision (mAP) measurement is a common way to describe the classification accuracy of object detectors such as Mask RCNN. The model trained in this experiment achieved a $mAP = 0:8937$ with a IOU threshold equal to 0.5 when tested on the synthetic validation dataset. This means that if the detected object's predicted mask does create a IOU with the ground truth mask of more than 0.5 the detection is immediately written off as wrong.

To gain further insights into the performance of the classifier network the confusion matrix shown in Figure 6.5 was created. Confusion matrices are explained further in Section 2.3.9. The ground truth classes are shown along the x-axis, while the prediction class probability distributions are shown along the y-axis. The figure was generated by testing the network on all 1500 synthetic validation images and logging the predicted classes. The IOU threshold was set equal to 0.5. During the evaluation the model produced on average 0.74 false positives per image. Note that non maximal suppression was not active during the test, so this value could probably have been reduced.

[1] Panoptic segmentation is equivalent to semantic segmentation but also assigns each pixel to an instance, not just a class.