Master's thesis

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

Andreas Thyholt Henriksen

Domain Adaptation for Maritime Instance Segmentation: From Synthetic Data to the Real-World

A Study on Generation and Use of Synthetic Data in Convolutional Neural Networks and on Model-Agnostic Explenations in Instance Segmentation

Master's thesis in Cybernetics and Robotics
Supervisor: Anastasios Lekkas
July 2019

NTNU
Norwegian University of Science and Technology

Summary

This thesis aims to explore practical implications from using a maritime simulator for quick generation of synthetic labelled images, with the purpose of training Convolutional Neural Networks (CNNs) for instance segmentation and classification in maritime scenarios. The motivation has been to contribute towards overcoming the challenges associated with building large real-world datasets due to the intrinsic difficulty of image acquisition and the time-consuming manual labelling effort, through the use of computer graphics. In addition, it aims to investigate if methods within the area of Explainable AI (XAI) can be used to obtain explanations from instance segmentation models.

First, the area of XAI is presented and a method is proposed for dedicating an existing XAI method named Local Interpretable Model-agnostic Explanations (LIME) for instance segmentation models for generating explanation. Later, the proposed method shows that LIME can be used to explain both object detection and mask prediction from an instance segmentation model

Second, a method is developed for acquiring synthetic images from a maritime simulator with instance-level labels of deployable objects, based on a solution with access only to the user interface and the model library, that is without access to the software. Two replays of the same simulation, and manipulation of texture between these two replays highlights the objects and serves as basis for the acquisition and labeling of images. Synchronized pairs of images is then obtained by screen capture from these two replays, with timing based on reading the internal log clock made visually available in the user interface using a Random Forest digital digit classifier. Object masks are subsequently extracted from images with manipulated texture using thresholding of weak background colors, morphological transformation, connected component calculation, probabilistic hierarchical clustering based on Variational Bayesian learning of Gaussian Mixture Models (VBGMM) and agglomerative hierarchical clustering, and subsequent manual assessment and adjustment. This method allows for further studying the implications of using synthetic data.

Last, an experiment is conducted based on a synthetic dataset, a limited sized real-world dataset, a Convolution Neural Network (CNN) named Mask R-CNN, domain adaptation techniques based on both three-stage gradual fine-tuning and full fine-tuning, and data augmentation. LIME is also part of this experiment. Six different models are trained; two pure synthetic models, two real-world adapted models based on the best performing synthetic model on the synthetic test-set, and two pure real-world control models. Results suggests that the use of synthetic images has a slight performance impact on all the adapted models and that they contained a greater portion of knowledge about cross-domain features than the corresponding control models with in terms of fine-tuning approaches. In the case of the pure domain models, gradual fine-tuning achieved better performance than full fine-tuning and was more capable of learning domain-invariant features. In the case of the adapted models, gradual fine-tuning and full fine-tuning achieved approximately equal performance. Furthermore, results indicate that learning of domain-invariant features continuous after stagnation in validation-loss during the training of synthetic models, suggesting that validation data from both the synthetic and real-world domain should be used when training of synthetic models.

Sammendrag

Denne avhandlingen tar sikte på å utforske praktiske implikasjoner ved bruk av en maritim simulator for rask generering av syntetiske instans-baserte sannhets-merkede bilder med det formål å trene modeller basert på konvolusjonelle nevrale nettverk (CNNs) for instanssegmentering og klassifisering av bilder i maritime scenarioer. Motivasjonen har vært å bidra til å overvinne utfordringene knyttet til å bygge store reelle datasett, grunnet den iboende vanskeligheten knyttet til anskaffe relevante bilder og det tidkrevende manuelle merkingsarbeidet, ved bruk av datagrafikk. I tillegg tar avhandlingen til sikte på å undersøke om metoder innenfor fag-området Explainable AI (XAI) kan brukes til å generere forklaringer for instans-segmenteringsmodeller.

Først presenteres et fagområde XAI. Deretter fremlegges et forslag for å dedikere en eksisterende XAI-metode, kalt Local Interpretable Model-agnostic Explanations (LIME), til instans-baserte segmenteringsmodeller for å generere forklaringer på prediksjoner. Senere blir LIME vist til å kunne gi forklaringer på både objekt-deteksjon og silhuett-prediksjon fra en instans-basert modell.

Deretter utvikles en metode for å skaffe syntetiske bilder fra en maritim simulator med instans-basert sannhets-merking av objekter, basert på en løsning med tilgang kun til brukergrensesnittet og modellbiblioteket. To opptak av samme simulering med manipulering av tekstur mellom disse to opptakene, fremhever objektene og fungerer i etterkant som grunnlag for anskaffelse og merking av bilder. Deretter tas par med synkroniserte skjermbilder, med synkronisering basert på intern logg-klokke i simulatoren gjort visuelt tilgjengelig i brukergrensesnittet og avlest automatisk ved hjelp av en Random Forest digital tall klassifikator. Instans-baserte sannhets-merker blir deretter generert basert på bilder med manipulert tekstur ved hjelp av filtrering av bakgrunns-farger, morfologiske transformasjoner, beregninger av tilkoblede komponenter, sannsynlighets-hierarkisk clustering basert på Variasjonel Bayesisk læring av Gaussiske komponent-modeller (VGBMM) og agglomerativ hierarkisk clustering, og etterfølgende manuell vurdering og justering. Denne metoden tillater for videre studier av implikasjonene ved bruk av syntetiske data.

Deretter utføres et eksperiment basert på et syntetisk datasett, et reelt datasett av liten størrelse, et konvolusjonelle nevrale nettverk kalt Mask R-CNN, domenetilpasningsteknikker basert på både tre-trinns gradvis finjustering og full finjustering, og data-augmentering. Seks forskjellige modeller trenes; To rene syntetiske modeller, to reelle domenetilpassede modeller basert på den beste syntetiske modellen på det syntetiske testsettet, og to rene reelle kontrollmodeller. Resultatene tyder på at bruken av syntetiske bilder har en merkbar påvirkning på prestasjonen til alle domenetilpassede modeller, og at de inneholdt en større del av kunnskaper på tvers av domenene sammenlignet med de korresponderende kontrollmodellene. Videre tyder resultatene fra de rene domenemodellene at gradvis finjustering oppnår en bedre prestasjon enn full finjustering, og var bedre i stand til å lære kunnskaper på tvers av domenene. For de tilpassede modellene oppnådde gradvis finjustering og full finjustering omtrent like stor prestasjon. Videre indikerer resultatene at læring av kunnskaper på tvers av domenene fortsetter etter stagnasjon i valideringstap under opplæring av syntetiske modeller, noe som tyder på at valideringsdata fra både det syntetiske og det reelle domenet bør brukes ved trening av syntetiske modeller.

Preface

This thesis is the culmination of my work carried out during the spring and early summer of 2019 at the Norwegian University of Science and Technology under the supervision of Anastasios Lekkas and in cooperation with Kongsberg Digital, and concludes my Master of Science in Cybernetics and Robotics in Autonomous Systems. Writing this thesis has been both inspiring and challenging, as it's about a technology that has experienced a renaissance in the past few years.

Kongsberg Digital has provided me the tools necessary to carry out the research in this thesis. In addition, multiple software libraries and online services were utilized. The following is a summary of all information and software used as a basis for the research, as well as the help I received during the work.

- A Dell Optiplex 9010 computer and a workplace was provided by NTNU.
- Kongsberg Digital provided a maritime simulator named K-Sim, and a model library with both vessel models and geographical area models.
- Kongsberg Digital provided a HP ZBook 15 computer with Windows 10 that was used to run K-Sim during the implementation and acquisition of synthetic images. Sigrid Fosen at Kongsberg Digital has taken care of access to the computer and IT security efforts.
- An open-source implementation of the instance-segmentation neural network architecture Mask R-CNN was obtained from github.com/matterport/Mask_RCNN, used as a basis for the experiment about instance segmentation.
- An open-source implementation of LIME was obtained from github.com/marcotcr/lime, and used as a basis for the experiment regarding instance segmentation and LIME.
- The OpenCV software library was used to perform image processing tasks.
- The Imgaug software library was used to perform image augmentation transformation.
- The Scikit-learn software library was used for building machine learners.
- The Scikit-image software library was used for image segmenting techniques.
- The Google Colaboratory free-of-charge cloud service was used for training of MASK R-CNN models and execution of LIME.
- The version-control system Git and hosting platform Github was used for version control and file transferring between the two computers and Google Colaboratory.

My supervisor has assisted me through fruitful discussions and providing feedback on my thesis. Pierluigi Salvo Rossi has been the industrial contact person at Kongsberg Digital, Pierluigi has provided for the collaboration with Kongsberg Digital. Industrial co-supervisor Thorvald Grindstad at Kongsberg has provided guidance in the use of K-Sim, and offered access to software and model libraries. Besides this and from the elements on the list above, my work has been independent in its nature.

Acknowledgement

I would first like to thank my supervisor Anastasios Lekkas of the Department of Engineering Cybernetics at NTNU. He has consistently allowed for this thesis to be my work and has helped me through frequent and fruitful discussions.

I would also like to thank industrial contact person Pierluigi Salvo Rossi for providing cooperation with Kongsberg Digital, industrial co-supervisor Thorvald Grindstad for providing access to Kongsberg Digital's maritime simulator and guidance in its use, and Kongsberg Digital for allowing for the cooperation.

Finally, I must express my profound gratitude to my family and my girlfriend, Stina Holen Friisø, for providing me with inspiration and unfailing support throughout my years of study and through this thesis.

22.07.2019
Andreas Thyholt Henriksen

Table of Contents

List of Tables

List of Figures

Nomenclature

AI	Artificial Intelligence. 33, 34, 107
ANN	Artificial Neural Network. ix, 17–19
BGMM	Bayesian Gaussian Mixture Models. 43
CART	Classification And Regression Trees. 7
CNN	Convolutional Neural Network. ix, 1–3, 20, 22, 23, 25, 27, 28
CV	Computer Vision. 1, 13
DA	Domain Adaption. 1, 28, 107
DL	Deep Learning. 1
DNN	Deep Neural Network. 1, 20
Fast R-CNN	A two-state region-based CNN architecture based on the R-CNN architecture, designed for solving the object detection problem. 13
Faster R-CNN	A two-state region-based CNN architecture based on the Fast R-CNN architecture, designed for solving the object detection problem. 13, 26, 27
FCN	Fully Convolutional Network. 26
FPN	Feature Pyramid Network. 26, 27, 62, 68, 72
GDPR	General Data Protection Regulation. 34
GMM	Gaussian Mixture Models. 9, 10, 53–55

GPU	Graphical Processing Unit. 1, 73
IoU	Intersection over Union. 15, 17, 64, 79–82, 85, 86, 88
LIME	Local Interpretable Model-agnostic Explanations. x, 4, 33, 35–37, 61, 64, 79–81, 83, 85, 92, 94, 95, 97, 104–107
mAP	mean Average Precision. 14–17, 24, 25, 63, 78, 85, 88, 101, 104, 106
Mask R-CNN	A two-state region-based CNN architecture based on the Faster R-CNN architecture, designed for solving the instance segmentation problem. ix, 3, 13, 24, 26, 27, 62–64, 68, 72–74, 78, 105, 107
ML	Machine Learning. 6
MS COCO	Microsoft COCO: Common Objects in Context. 63, 72, 73, 78, 106
R-CNN	A two-state region-based method which runs a CNN feature extractor and a SVM classifier multiple on region proposals from Selective Search for solving the object detection problem. 13, 24, 25
Resnet101	ResNet-101 is a residual convolutional neural network that is 101 layers deep, trained on more than a million images from the ImageNet database [1], and capable of classifying images into 1000 object categories. 24, 26, 62, 63, 72
SGD	Stochastic Gradient Descent. 19
SSD	Single-Shot Detector, a single-state CNN architecture designed for solving object detection problem. 13
SVM	Support-Vector Machine. 53
TL	Transfer Learning. 28
TPU	Tensor Processing Unit. 73
VBGMM	Variational Bayesian Gaussian Mixture Models. ix, 10, 11, 53–55, 105, 106
VIA	VGG Image Annotator, a stand-alone manual annotation software for image, audio and video by [2, 3]. 62
XAI	Explainable AI. 2–4, 33, 34, 37, 105, 107
YOLO	You Only Look Once, a single-state CNN architecture designed for solving object detection problem. 13

Chapter 1

Introduction

1.1 Background

In recent years, Convolutional Neural Networks (CNNs) has within the field of Computer Vision achieved amazing performances in various tasks, e.g. image classification, object detection, and image segmentation. In particular, Deep Learning (DL) have in combination with high flexibility of software frameworks, large public image repositories, and especially the advancement in GPU-accelerated computation techniques played an essential role. Unfortunately, today's deep neural networks require large amounts of data to generalize well beyond the data used in training due to their massive size in terms of trainable model parameters, often exceeding the number of samples they are trained on. The process of acquisition, processing and labelling of customized data may be time-consuming and costly, due to the intrinsic difficulty of image acquisition or substantial demand for manual labelling effort. Recently, computer graphics have drawn the attention of academia and the industry as a way of automating this process by generating synthetic training data, to be used as a substitute when real-world data is unavailable, scarce or limited in diversity. Domain Adaption techniques enables the use through fine tuning, layer freezing, data augmentation, and mixing a synthetic and real-world data. However, synthetic data meant to imitate the real-world suffers from a covariate shift as the distribution of features changes between the domains. Due to the poorly understood nature of "black-box" Deep Neural Networks (DNNs), particularly in terms of generalization and transferability of features [4, 5], it is hard to tell how a transition between synthetic and real-world data affects these machine learners.

Kongsberg Digital has proposed to use their maritime simulator, named K-Sim, to explore, understand and summarize practical implications of using computer graphics for quick generation of useful labelled images, for image segmentation and classification in the maritime domain. K-Sim offers highly realistic maritime scenarios in multiple geographical areas and all possible weather conditions, with a wide variety of vessels and

objects, and is used for maritime education, training and studies. An advanced physical engine and state-of-the-art hydrodynamic modelling bring the maritime scenarios close to the real world. Therefore, it would be interesting to gain insight into the implication of using synthetic data acquired from K-Sim for training various CNN for later deployment into the real-world maritime domain, in terms of acquisition and labeling, domain adaptation, and performance influence. The benefits from using synthetic images and data from K-Sim are many, as the real-world maritime domain has limited accessibility both visual and informational, due to the vast number of different types of vessels, scenarios, environments, and the fact the open ocean naturally demands time and resources to access. K-Sim could be used to create pure synthetic image datasets, to balance unbalanced real-world dataset, or create datasets from specific scenarios that may be practically inaccessible in the real world. Besides, since K-Sim is developed for maritime education and training, there are reasons to believe that it can contribute to the development of autonomous maritime systems by providing it's virtual maritime environment.

Recently, the implications of the "black-box" nature of machine learning models have gained attention from the community and academia, who ask questions regarding trust and explanations in connection with societal, moral, legal norms, and regulations, and which has lead to a new paradigm within AI, called Explainable AI (XAI). It is believed that machine learning models that offer explainability cannot only assert trust and meet legal regulations, but also help scientists and developers to reveal flaws, bottlenecks, and opportunities for improvement in the model design. It might also lead to more principled architecture design. Therefore, it would also be interesting to explore if recent methods within the area of XAI can be used for obtaining insight into image segmentation models.

Figure 1.1: K-Sim graphic illustration. Source: [6]

1.2 Problem description

The problem that this thesis aims to answer is summarized as:

> *The main objectives in this thesis is to explore, understand, and summarize the practical implication of using a maritime simulator for quick generation of "useful" labelled images, for instance segmentation and classification in maritime scenarios. The study is based on the use of the Kongsberg Digital Maritime Simulator for computer-generated synthetic training data, Mask R-CNN for image segmentation and classification, and different domain adaption techniques for fine tuning of networks. In addition, to the extent possible, the interpretability of the models developed in this thesis is investigated using recent methods within the area of XAI.*

1.3 Objectives

The objectives based on the problem description in the previous section have been divided into four main parts:

1. Conduct a short study on the topic of Explainable AI, and investigate if there is a method that can be used for obtaining explanation from instance segmentation models.
2. Develop and implement a method for generation of instance-level labelled images from K-Sim. The method will be based on K-Sim's user interface, access to the model library, and manipulation of graphics followed by a mask extraction step.
3. Conduct an experiment based on domain adaptation with different fine-tuning approaches, appropriate image augmentation, synthetic images from K-Sim as source data, and real-world images as target data. We will use Mask R-CNN as the CNN architecture.
4. Attempt to obtain insight or explanations of detections from an instance segmentation model using the proposed Explainable AI method.

1.4 Outline

This thesis is organized into mainly five parts:

1. Theoretical background and Explainable AI.
2. Development, implementation, and evaluation of a method for acquisition of instance-level labelled synthetic data from K-Sim

3. Domain adaptation and LIME experiment.

4. Results, discussion, conclusion, and future work.

The chapter has been organized with respect to these parts. The theoretical background in Chapter 2 serves as the theoretical foundation for the rest of the thesis, and has been written with the novice reader in mind. It starts by covering the basic paradigms within machine learning before it covers in details the tools that have been utilized throughout the thesis.

Chapter 3 presents a brief introduction to the topic of Explainable AI in the context of computer vision, by giving the motivation and proposing the use of a method called LIME for obtaining explanations from an instance segmentation model.

Chapter 4 introduces Kongsberg Digital's maritime simulator called K-Sim, before covering in details the method developed for acquiring instance-level labelled synthetic images. It is based on the manipulation of the simulator's graphics and a method which can be described as probabilistic hierarchical clustering based on the hue-dimension of the dominant colors and spatially closely connected regions. We delay a discussion of this method until the discussion.

Chapter 5 conducts an experiment based on domain adaptation with different fine-tuning approaches, appropriate image augmentation, synthetic images from K-Sim as source data, and real-world images as target data. Based on the best performing adapted model, LIME is used to obtain an explanation for two different detections. The chapter ends with a brief discussion and evaluation of the method and proposals for improvements.

Chapter 6 presents the results from the experiment along with a few performance inspection tools to shed light on the performance and differences.

Chapter 7 discusses the results from the experiment, compares the different domain adaptation techniques, and gives suggestions to how the performance or results could have been improved. Also, it discusses the method developed for acquisition of instance-level labeled images from K-Sim in terms of what it can offer and its limitations.

Lastly and finally, Chapter 8 wraps up the knowledge obtained throughout the thesis and reaches a conclusion. The thesis ends with proposals for further work on the topics.

Chapter 2

Theoretical background

This chapter presents the relevant terminology and the many theories, methods, and concepts that form the theoretical and technological foundation of this thesis, and has dived them into six parts:

- Section 2.1 introduces the field of machine learning.
- Section 2.2 cover an ensemble classification approach named Random Forests Classification
- Section 2.3 covers two distinct methods for clustering; variational learning for Gaussian Mixture Models and agglomerative hierarchical clustering.
- Section 2.4 covers the technology behind image segmentation. It first clarifies what separates the task of image and instance segmentation from other tasks within computer vision then covers the concept of artificial -and convolutional neural networks, and lastly presents different neural network architectures that have laid the road towards state-of-the-art performance within the task of instance segmentation.
- Section 2.5 covers the field of transfer learning and domain adaptation, and different types of dataset biases that can affect performance.
- Section 2.6 covers synthetic data for visual applications.

The competent reader can expect to find the content of these sections familiar. Nevertheless, it is encouraged to read Section 2.5 since it states related terminology to domain adaptation used throughout the thesis.

2.1 Machine Learning

Machine Learning (ML) is a discipline within computer science that is concerned about algorithms and statistical models that can accomplish the task effectively by relying on inference and learning from given data. The core idea is that knowledge is learned from experience after having witnessed the outcome from a decision taken based on input data. Models that are capable of learning are denoted as learners throughout this thesis. There are mainly three machine different learning paradigms:

- Supervised learning
- Unsupervised learning
- Reinforcement learning

In supervised learning, a training set is given to the learner that contains pairs of input and output, denoted as labelled data points or data points with ground truth. The learner then attempts to learn an inferred function which can map new input. In other words, the learner attempts to learn the relationship between the input and output, such that it can predict the correct output for new unseen input datasets. The full-relationship between input and output are called the target function, which predicts with 100 % accuracy. In unsupervised learning, a learner is instead given a training set without ground truths and has to find unknown patterns. Thus the problem at hand becomes what is called pattern recognition, that is categorization without supervision. Lastly, in reinforcement learning, the learner or actor is deployed into unknown environments and ought to take actions to maximize some notion of cumulative reward. In layman's term, the learner searches for ways to accomplish a set of goals by taking actions in the environment and receive occasional rewards, as in an escape-room fashion. The methods implemented in this thesis exclusively base themselves on supervised and unsupervised learning.

2.2 Random Forests Classifier

Random Forests Classifier or Random Forests is a supervised ensemble method that exclusively uses trees as base learners. A digital digit classifier in Section 4.2.2 used to read time from a digital screen used a random forest classifier. We shall explain the method by cutting it down to its basics, by first looking at a single decision tree, then bagged decision trees, and finally, Random forest.

Decision Tree

Decision tree is a category of learners that solves supervised tasks using only conditional split statements represented by a tree structure. A tree is constructed as a connection of nodes, which starts at the root node, splits into subsequent branches, and end at the terminal branches or leaves. These learners take a set of different sorts of values or attributes as input, splits on one of the given attributes per node into the tree in an iterative fashion, and

terminate at the leaves where it makes a decision. The attributes may be binary, numerical, strings, or any other data type. A decision tree is illustrated in Figure 2.1.

The challenge is how to build the tree that it how to determine the tree structure of conditional split statements. If it is desired to minimize the depth of the tree, it is common to use an algorithm called ID3 after J.R. Quinland[7] which employed a top-down, greedy search through the space of possible splits using entropy and information gain. The entropy is a measure of uncertainty of a set after C.E. Shannon and can be understood intuitively as the length of the encoding necessary to represent a set based on the attribute frequencies. Entropy is for a given sample of sets given by:

$$H(S) = \sum_{i=1}^{C} -p_i \log_2 p_i \tag{2.1}$$

where C is a set of attributes, S is the sample of sets, and p_i is the relative attribute frequency of S. The information gain is based on the decrease in entropy after splitting on an attribute, which entails the removal of that attribute from the set. Information gain is given by:

$$I(S, A) = H(S) - \sum_{\theta \in A} \frac{|S_\theta|}{|S|} \cdot H\left(S_\theta\right) \tag{2.2}$$

$$= H(S) - H(S|A), \tag{2.3}$$

where A is the attribute that splits the set S, θ is each value or range of values of A, and S_θ is the subsets with equal values of A. Information gain can be understood intuitively as the reduction in entropy given an additional piece of information A about S. Hence, the attribute that gives the greatest entropy reduction is chosen as the condition variable for the split.

The procedure of splitting is repeated until reaching a full dataset splitting or maximum depth. The maximum depth hyper-parameter is often used to avoid overfitting of the decision three, that is a too closely fit to a limited set of datapoints which may result in a failure to predict future datapoints.

The implementation of the decision tree used in this thesis is obtained from scikit-learn[8], and is based on an algorithm called Classification And Regression Trees (CART) which constructs binary trees. CART is a variant of a decision-tree algorithm named C4.5 which is a widely used algorithm, which in turn is a successor of the ID3 algorithm.

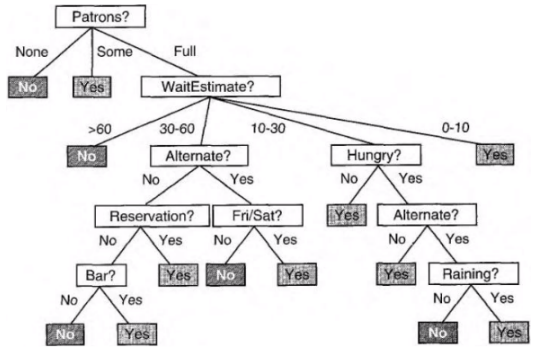


Figure 2.1: An illustration of an decision tree for deciding whether to wait for a table. Source: [9].

Bagged Tree ensemble

Ensemble learning or combined predictors is a powerful machine learning paradigm which combines multiple weak learners to solve a task. Unlike general base learners, which learns a single model from data, ensemble methods combine predictions from multiple base learners to improve performance. Decision trees have high variance and low bias, since it is highly susceptible to variances in data but make almost no assumption about the target function.

Bagged trees use a technique called bagging or bootstrapping aggregating, were many decision trees are trained on different training set sampled randomly with replacement and combined using the average of predictions. The result is a reduction in the variance due to averaging of predictions compared to a single decision tree, but at the cost of an increased bias since there is less data for each decision tree to learn from. If the output of the decision trees is probabilities, it is also possible if not beneficial to average over the probabilities rather than averaging over binary class prediction, for example.

Random Forest Classifier

Random Forest takes one extra step from bagging where in addition to sampling subsets at random, it also selects a subset of attributes at random rather than using all attributes. There are several different benefits from this approach:

- It results in a moderate variance compared to the high variance for single decision trees, while keeping the bias low.
- It decorrelates the trees, which is helpful in case there is a big difference in the predictive ability among the attributes.
- It contributes towards handling higher dimensional data since each tree only considers a subset of attributes

- It handles unbalanced data by allowing for higher error rate for small classes compared to larger classes.
- It allows for parallel computation.

However, random forest learners are not very interpretable and acts as a "black box", they can take up a large amount of memory, and they tend to overfit. due to Due to the risk of overfitting, it is necessary to tune the hyper-parameters. A list of important and useful hyper-parameters is given below:

- The number of estimators which is the number of trees in the forest.
- The maximum number of features when considering the best split
- The maximum depth, that is the maximum number of successive splits.
- The class weight, in case that class imbalance requires compensation.

2.3 Probabilistic and Hierachical clustering

Clustering or clustering analysis is a type of unsupervised learning method within machine learning that is used to find clusters or groups inherent in a set of datapoints. Based on a measure of similarity that serves a specific purpose, datapoints in the same cluster should be more similar to each other than to those in other groups. Two different types of unsupervised clustering algorithms are Variational Bayesian Gaussian Mixture Models and agglomerative weighted hierarchical clustering. The following two sections will give an insight into these two algorithms. However, since the content of this thesis does not depend on understanding the algorithm's technical and statistical complexity, we stick to the main concepts along with strengths and weaknesses. We refer to the sources from which the content of these two section are based, and advise the inquisitive reader to seek more information there. Section 4.2.3 further combine them into a two-stage probabilistic hierarchical clustering method for obtaining non-Gaussian clusters out of a large collection of datapoints within reasonable time.

2.3.1 Variational Bayesian Gaussian Mixture Models

The content of this section is based on [10, 11].

Gaussian Mixture Models (GMM) are probabilistic representations for multimodal distributions and are based on a linear combination of unimodal Gaussian component distributions. GMM maintain many of the theoretical and computational benefits of Gaussian models, making them convenient for efficiently modelling huge amount of data. Clustering can be implemented based on a GMM by choosing the most likely component assignment using Bayes' theorem. Given that the numbers of components are unknown, the learning task consists of estimating the number of components, as well as estimating the parameters that characterize the distributions. Conventional approaches for finding the number of

components based on exhaustive cross-validation in the number of components up to maximum value are computationally expensive, are wasteful of data, and give noisy estimates. Instead, it is possible to use variational Bayesian methods for estimating the number of components as well as estimating the parameters. We denote the inference algorithm of learning a GMM using Variational Bayes as Variational Bayesian Gaussian Mixture Models (VBGMM). The following subsection covers the two main aspects of this inference algorithm, first the GMM and then Variational Bayes. The subsection gives the reader an idea of the principals of the algorithm.

Gaussian Mixture Models and clustering

A GMM is defined by the following equation:

$$p(\mathbf{x}) = \sum_{i=1}^{K} \pi_i \mathcal{N}\left(\mathbf{x}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\right), \tag{2.4}$$

where π_k are the mixing coefficients who satisfies $0 \leq \pi_i \leq 1$ and $\sum_{i=1}^{M} \pi_i = 1$, μ_i are the means, $\sum_i$ are the covariance matrix, and $\mathcal{N}\left(\mathbf{x}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\right)$ are the multivariate Gaussian distribution.

Each datapoint $\boldsymbol{x}_n$ also have an associated latent one-to-K binary vector $\boldsymbol{c}_n$ of length K, which indicates which component distribution $\boldsymbol{x}_n$ originates from. If $\boldsymbol{x}_n$ originates from component k, then $\boldsymbol{c}_k$ is equal to one, while all other elements are equal to zero. Given a component c_k and a GMM, the posterior component distribution assignment probability can be calculated using Bayes' theorem:

$$p\left(c_k|\boldsymbol{x}\right) = \frac{p\left(\boldsymbol{x}, c_k\right)}{p(\boldsymbol{x})} = \frac{p\left(c_k\right) p\left(\boldsymbol{x}|c_k\right)}{\sum_{j=1}^{K} p\left(c_j\right) p\left(\boldsymbol{x}|c_j\right)} = \frac{\phi_k \mathcal{N}\left(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\right)}{\sum_{j=1}^{K} \phi_j \mathcal{N}\left(\boldsymbol{x}|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j\right)} \tag{2.5}$$

Choosing the most likely component assignment provides a way to learn clusters.

Variational Bayes estimation

The learning task consists of estimating the number of components and the parameters that characterize the distributions.

Bayesian statistics frames all inference about unknown quantities as calculation about the posterior, meaning that posterior knowledge about unknown quantities is obtained based on prior knowledge and new evidence using Bayes' theorem. Given a set of observed variables $\boldsymbol{x} = \boldsymbol{x}_{1:n}$ and a set of unobserved variables $\boldsymbol{z} = \boldsymbol{z}_{1:n}$, a conditional distribution based on Bayes' theorem can be written as:

$$p(\mathbf{z}|\mathbf{x}) = \frac{p(\mathbf{z}, \mathbf{x})}{p(\mathbf{x})} = \frac{p(\mathbf{z}, \mathbf{x})}{\int p(\mathbf{z}, \mathbf{x}) \mathrm{d}\mathbf{z}}. \tag{2.6}$$

However, the denominator or evidence integral is often unavailable in closed form or in-

tractable as it often requires exponential time to compute. Variational inference is used to approximate posterior distribution for Bayesian models by using a variational distribution:

$$p(\mathbf{z}|\mathbf{x}) \approx q(\mathbf{z}). \tag{2.7}$$

The variational distribution is found by minimizing a dissimilarity function given the following optimization problem:

$$q^*(\mathbf{z}) = \underset{q(\mathbf{z})\in\mathscr{Q}}{\arg\min}\, \mathrm{KL}(q(\mathbf{z})\|p(\mathbf{z}|\mathbf{x})), \tag{2.8}$$

where KL is the probability distribution dissimilarity function named Kullback-Leibler divergence, and $\mathscr{Q}$ is a variational family of distribution over the unobserved variables. Once solved, $q^*(\mathbf{z})$ is the best approximation of the conditional distribution, within the variational family of $\mathscr{Q}$. $\mathrm{KL}(q||p)$ is given by:

$$\mathrm{KL}(q(\mathbf{z})\|p(\mathbf{z}|\mathbf{x})) = \mathbb{E}[\log q(\mathbf{z})] - \mathbb{E}[\log p(\mathbf{z},\mathbf{x})] + \log p(\mathbf{x}), \tag{2.9}$$

but is not computable due to the dependency on $\log p(\mathbf{x})$. However, since $p(\mathbf{x})$ is constant, maximizing the negative of the first two terms, called the evidence lower bound (ELBO):

$$\mathrm{ELBO}(q) = \mathbb{E}[\log p(\mathbf{z},\mathbf{x})] - \mathbb{E}[\log q(\mathbf{z})], \tag{2.10}$$

is equivalent to minimizing the KL divergence. The complexity of $\mathscr{Q}$ determines the complexity of the optimization; it is more complex to optimize a complex family than a simple family.

The mean-field variational family defines a specific class of joint distributions that are based on the assumption of mutually independent unobserved variables that are governed by distinct factors in the variational distribution. The factorization is given by:

$$q(\mathbf{z}) = \prod_{i=1}^{m} q_j\left(z_j\right). \tag{2.11}$$

In the optimization, these variational factors are chosen to maximize the evidence lower bound in Eq. 2.10 using variational calculus. VBGMM is obtained by choosing one factor for each latent variable in the Gaussian mixture model.

2.3.2 Agglomerative Weighted Hierarchical Clustering

Hierarchical clustering is a method which seeks to build a hierarchy of clusters. There are two types of hierarchical clustering, namely divisive and agglomerative. The former is a top-down clustering method where we start with a single cluster and then partition the cluster to two least similar clusters based on a dissimilarity measure. We are concerned about the latter, which is a bottom-up clustering method where we start with as many clusters as datapoints, and combines in an iterative fashion the two most similar clusters based

on a similarity measure. This iterative fashion proceeds until there is only one cluster left, yielding a hierarchical structure of the clusters called a dendrogram. The data is clustered by implementing a threshold for the maximum allowed distance between clusters, which divides the dendrogram into final distinct clusters. This method is illustrated in Figure 2.2, where ten datapoints with values from 8 to 85 are clustered into 4 clusters using a threshold equal to 10.

There are various similarity measures or linkage methods, such as a single linkage which considers the shortest distance between two points in each cluster, and complete linkage which considers the longest distance between two points in each cluster. The similarity measure used in this thesis is based on what can be described as a weighted pairwise distance measure. After having combined the two nearest cluster, say s and t, into a single cluster v, its distance to a new cluster u is given by:

$$d(u, v) = (\text{dist}(s, v) + \text{dist}(t, v))/2. \qquad (2.12)$$

where $\text{dist}(\dots)$ is a distance measure, e.g. Manhattan metric.

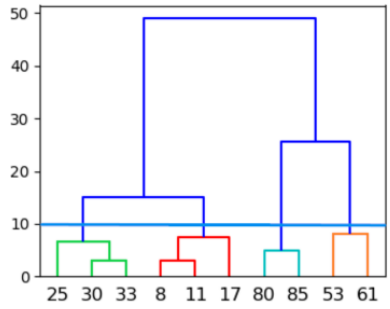


Figure 2.2: Illustration of agglomerative weighted hierarchical Clustering

2.4 Image segmentation

Image segmentation is a sub-field within image recognition, and addresses the problem of partitioning an digital image or video into multiple segments or set of pixels. This section covers in details the road from the classical task of image classification towards the task if instance segmentation, which this thesis is based, covering the terminology, concepts, and technology that lies behind it. We start by explaining different tasks within image segmentation and clarify what separates instance segmentation from other tasks. We then proceed with covering the concepts of artificial and convolutions neural networks and present different neural network architectures. Note that these architectures are referred to from the beginning in the context of the different tasks.

By the end of this section, the reader should have a clear view of what image and instance segmentation is and how it deep learning solves these tasks.

2.4.1 Main tasks within image recognition

Image recognition in the context of single two-dimensional images may roughly be divided into four main tasks, as shown in Figure 2.3. Note that they are not isolated fields but rather natural steps in the progression from coarse to finer inference and that there are other tasks as well.

Image Classification

Image Classification is the task of assigning an image one label from a fixed set of categories. It is common practice to output a probability prediction distributed over all classes representing the classification confidence that an image belongs to certain classes. Because many other tasks within image recognition such as object detection involve the task of image classification, it has a large variety of practical applications and is considered one of the core problems in Computer Vision. For instance, the problem of handwritten digit classification is a well-studied image classification problem. Today, deep neural networks and convolutional neural networks, in particular, has been widely adopted for this task.

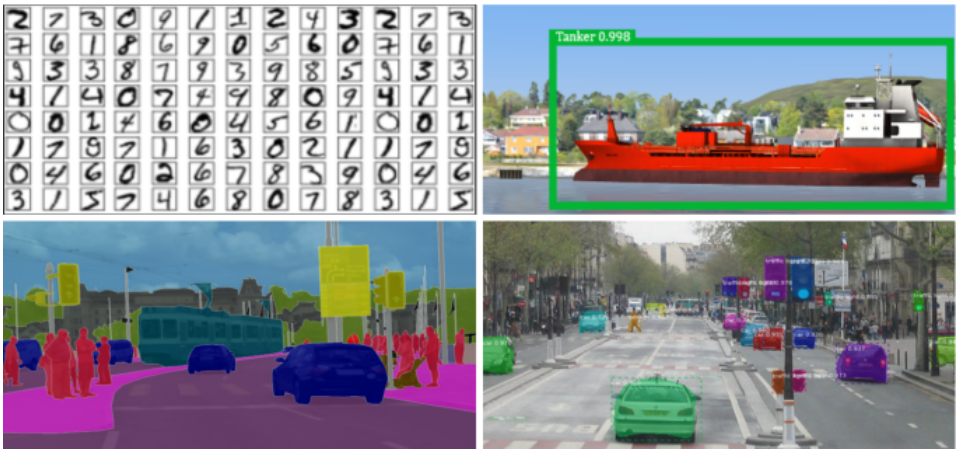


Figure 2.3: Illustration of four different objectives within image recognition. Upper left: image handwritten digit classification, source: Mnist handwritten dataser). Upper right: object detection. Lower left: semantic segmentation, source: Cityscrapes dataset. Lower right: instance segmentation, source: Matterport[12]

Object detection

Object detection is the task of locating and classifying semantically meaningful objects of a certain class in an image, and usually involves encapsulation of each object by a *bounding box* together with an assigned *class label* and a detection confidence score similar to that of image classification. A comprehensive survey of recent achievements in the field of object detection brought by deep learning techniques is given by [13]. Here it is noted that one of the factors for the tremendous success in object detection has been the development of better detection frameworks, mainly two-state region-based (R-CNN [14], Fast R-CNN [15], Faster R-CNN [16], and Mask R-CNN [17]) and one-state detectors (YOLO [18] and SSD [19]). While the best accuracy is obtained by region-based detection frameworks, they are too computationally intensive for embedded or real-time systems. Contrary, single-state frameworks is simpler and faster and achieves real-time detection, but they have not reached the same accuracy as region-based architectures.

Semantic segmentation

Semantic segmentation or scene segmentation can be described as a per-pixel labelling problem, concerned about labelling each pixel in an image or scene with the class of its enclosing object or region. Thus each pixel is assigned with a label based on its semantic meaning in the picture. Many different sophisticated neural network architectures proposed for this task have had great success, which is beyond the topic of this thesis. However, they make use of a similar neural network backbone such as residual neural networks and feature pyramid networks. We shall explain these techniques and neural networks in the upcoming sections.

Instance segmentation

Instance segmentation is the task of localizing and predicting class label and pixel-wise *mask* together with detection confidence to a varying number of instances present in an image. Besides, a bounding box can be also be predicted or given implicitly by the object mask. Thus, instance segmentation can also be considered a type of object detection task. The instances can be larger objects such as cars, pedestrians, and roads, or smaller parts of larger objects such as wheels of cars and bikes, and bags and hats on people. The task is seen as one of the most essential computer vision tasks. For instance, it is expected to contribute to the development of autonomous systems. Nonetheless, it is also one of the most challenging tasks as overlapping objects of the same class must be separated. Proposed in 2017, Mask R-CNN has in the recent years been among the state-of-the instance segmentation architectures [17] and is characterized as a simple and effective system. Today, it still serves as a basis for further development of architectures within the task of instance segmentation.

2.4.2 Performance measures

Before we dive into the neural network paradigm, we need to cover a popular metric for measuring accuracy within object detection and instance segmentation. The mean Average Precision (mAP) accuracy metric offers a unified way of comparing machine learners across different experiments, and takes into account a trade-off between what is called *precision* and *recall*, and potential class imbalance which could make accuracy metrics biased. As mentioned in the previous section, each of the tasks usually involves outputting a confidence score for each detection. The inherent problem which occurs is a what threshold should a detection be considered a valid detection. Lower thresholds yield more valid detections but with lesser confidence, resulting in a lower percentage of correct detections but higher recall; a fraction of actually detected object instances compared to the total amount of object instances. On the contrary, higher thresholds yield fewer detections but with higher confidence, resulting in a lower recall but a higher precision; a fraction of actually detected object instances among the total number of object detections. Thus, there is an inverse relationship between precision and recall and that these metrics are dependent on the model score threshold. Furthermore, class imbalance could make accuracy metrics

biased in case of a dominant class in terms of number demands a relatively high model capacity, archives high accuracy, and consequently dominates the accuracy matrices.

In mAP, all detections are kept to keep maximum information about a model while simultaneously handling the precision-recall trade-off and class unbalance. The following section describes the metric in details in the context of object detection and instance segmentation.

Detection conditions

The terminology related to the condition or correction of detections from a model in object detection is as follows:

- True-Positive (TP): A detection where the model correctly predicts the localization and the class.
- False-Positive (FP): A detection where the model incorrectly predicts the localization or/and the class.
- False Negative (FN): A detection where the model failed to predict the localization and class of a present object.
- True Negative (FN): Does not apply. The model did correctly not predict any presence of an object.

This terminology also applies to other machine learning tasks as well.

Intersection of Union

The Intersection over Union (IoU), or Jaccard similarity coefficient after the frenchman Paul Jaccard, measures the similarity between finite sample sets and is defined as the intersection divided by the union. In object detection and instance segmentation, a detection is considered a "correct match" if IoU between the bounding box or the instance mask and the ground truth segment is greater than a certain threshold, and is given by:

$$TP(Pr) = \frac{Pr \cap Gt}{Pr \cup Gt} > T, \tag{2.13}$$

where TP(Pr) is True-Positive given the prediction segment Pr and the ground truth segment Gt, and T is the IoU threshold. In case of instance segmentation where the predicted segment is pixel-wise masks, summation of pixels are used. It is common to vary the threshold T, depending on the requirements for considering a detection a correct match. Low values for T will allow inadequate segment to be accepted, while high values will impose strict requirements for accuracy.

Precision and Recall

Precision is a model's ability to identify only relevant object instances. It is the percentage of true-positive detections among all relevant detections and is given by:

$$\textit{Precision} = \frac{TP}{TP + FP} = \frac{TP}{n}, \tag{2.14}$$

where n is the total number of detections.

Recall is a model's ability to identify all relevant object instances. It is the percentage of true-positive detections among all relevant object ground truth instances and is given by:

$$\textit{Precision} = \frac{TP}{TP + FN} = \frac{TP}{m}, \tag{2.15}$$

where m is the total number of ground truth instances.

The inclusion of the notion *relevant* is in case there are multiple classes in the data points. As we see in the section after the next, mAP is based on an independent calculation of precision and recall between the classes.

Precision-Recall curve

The precision-recall curve is based on calculating precision and recall from most confident relevant detection and then towards the least confident relevant detection. Beforehand, the total number of ground truth segments is counted to get m as to calculate the recall. The corresponding precision and recall pairs are then calculated based on the accumulating number of true-positives and false-positives as one moves through the sorted detections. The precision-recall curve is finally given by the pairs of precision and recall values and limited to the quadrant given by the minimum value of zero and maximum value of one for both precision and recall.

Mean Average Precision

mean Average Precision (mAP) is based on calculating the mean area under the different precision-recall curves from each class over the whole dataset. The mathematical definition is:

$$\text{mAP} = \frac{1}{C} \sum_{j}^{C} \int_{0}^{1} p_j(r) dr \tag{2.16}$$

where C is the number of classes present in the dataset, and $p_j(r)$ is precision given recall r. However, the area under the precision-recall curve can be difficult to calculate. Therefore, we have in this thesis used a technique based on interpolation through all point in such a way that:

$$\text{mAP} = \frac{1}{C}\sum_{j}^{C}\sum_{r=0}^{1}(r_{n+1} - r_1)\tilde{\rho}_{interp,j}(r_{n+1}) \tag{2.17}$$

with:

$$\rho_{interp,j}(r_{n+1}) = \max_{\tilde{r} \geq r_{n+1}} \rho(\tilde{r}) \tag{2.18}$$

An illustration of precision-recall curves and calculations of average precisions is given in Figure 2.4. The area under the precision-recall curve is a good way to evaluate the accuracy of object detections and instance segmentations as confidence thresholds would not increase performance but rather possibly decrease it. Furthermore, mAP is a good way to evaluate performance for a dataset with class imbalance, since the maximum area under the curve is limited to one, thus mitigating the effect of biased accuracy. Since true-positives are based on the IoU-threshold, a common notation is mAP@T for telling that mAP is based a certain threshold. mAP is also extended by averaging over different thresholds, denoted as mAP@$[T_1...T_n]$. The results Section 6 from the experiment in Section 5 utilized both regular mAP and averaged mAP to provide good insight into the test results.

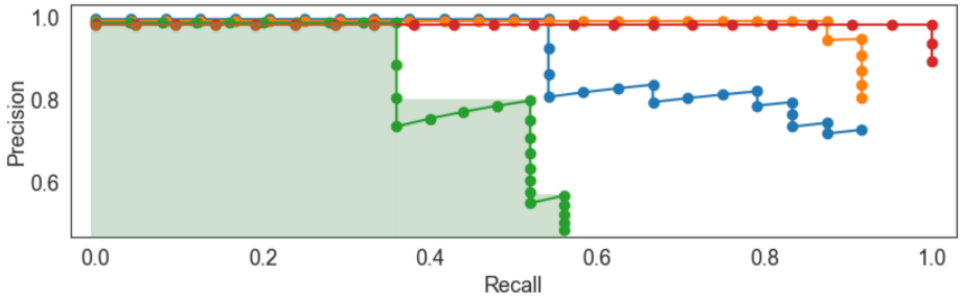


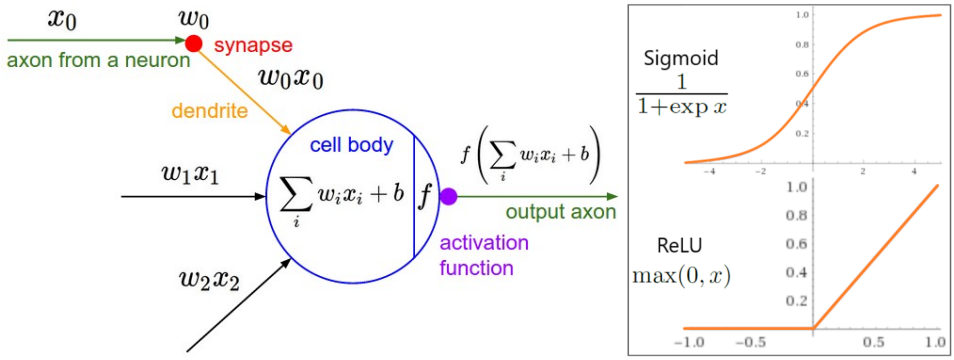
Figure 2.4: Illustration of precision-recall curves and calculation of average precision (green area). mAP is later calculated by taking the mean of the average precisions.

2.4.3 Artificial Neural Networks

Artificial Neural Networks (ANNs) denotes a computational tool that is inspired by and modelled after biological neural systems, by the use of simplified mathematical models of how the neurons in the human brain operate. Composing these simple models into a large graph or network where layers of nodes are connected by directed edges gives massive modelling power and flexibility.

Each node which is denoted as a *neuron* consist of a linear transformation of input with adjustable weights, a bias input, and a non-linear piece-wise continuous thresholding function denoted as *activation function*. The adjustable weights enable for learning, the bias allows for shifting the activation function, and the activation function allows for non-linear mapping and continuous output. The weights and bias are often denoted as the parame-

ters $\boldsymbol{\theta}$ of the neuron. There are a variety of different activation functions with different non-linear characteristics. It is each of these neurons that process the information through scaling, shifting and non-linear mapping and thresholding as it flows through the neural networks.



(a) Artificial neuron. Source: [20]

(b) Activation functions

Figure 2.5: Artificial neuron inspired by a biological neuron.

The neurons are organized in *layers*, where directed edges connect neuron in one layer to neuron in another subsequent layer. A layer can be represented mathematically by an unit bias vector $\boldsymbol{b}$, a weighting matrix $\boldsymbol{W}$, and an activation function $\sigma(\cdot)$. Given an input vector $\boldsymbol{x}$, the output vector $\boldsymbol{y}$ is given by

$$\boldsymbol{y} = \sigma(\mathbf{W}\boldsymbol{x} + \boldsymbol{b}), \tag{2.19}$$

where the activation function is applied to each of the elements of the vector from the linear transform. If all neuron in two adjacent layers is fully pair-wise connected, but the neurons within the single layers are not connected, those two layers are called *fully-connected layers*. A composition of layers constitutes a ANN, which is illustrated in Figure 2.6. During what is called the forward pass, inputs are propagated through the network. No cycles are allowed to happen as it would imply an infinite loop, and consequentially the network behaves like an acyclic graph and is called feed-forward neural network.

A loss function measures the goodness of fit between the neural network and the desired relationship between the input data and the ground truth. However, the choice of loss function depends on the objective. Within classification where outputs are probabilities and predict class membership, the cross-entropy cost function is the basis for many loss functions and is given by:

$$C = -\frac{1}{n}\sum_x \sum_i \left[z_i \ln y_i + (1 - z_i)\ln\left(1 - y_i\right)\right], \tag{2.20}$$

where i is the number of output neurons, y_i is the output of neuron i, z_i is the desired ground truth of neuron i, x is the input to the network, n is the total number of training instances, and $\ln$ is the natural logarithm with base 2. The natural logarithms is used due

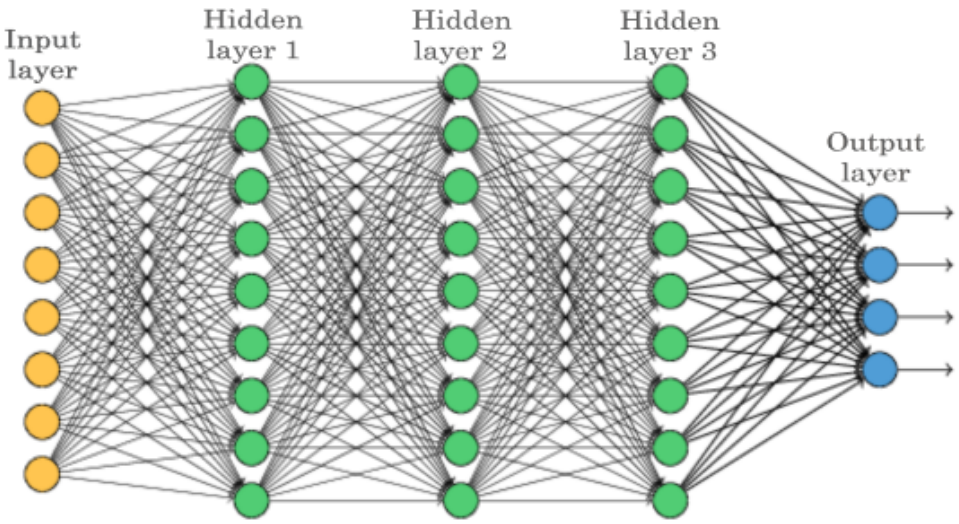


Figure 2.6: An illustration of an ANN consisting of an input layer with eight inputs, three hidden layers with nine neurons each, and one output layer with four neurons. Source: [21] (*Note: text have been changed and colors have been added to the image.*)

to the ease of calculating the derivatives, among other reasons. The reason for using cross-entropy is mainly for avoiding a slowdown in learning in case of a significant loss where activation functions are saturated. The reason for having the second term in the equation is to add a penalty for false-positives. Because ground truth class membership is binary, a false-negative would result in losing the sum over i if it was not for this term.

The challenge is how to determine the parameterization which best fits the neural network to the loss function. Because of the structured characteristics of neural network architecture, optimization of the loss function with respect to the parameters $\boldsymbol{\theta}$ offers a supervised learning scheme. A standard method for implementing this is *backpropagation*, short for "backwards propagation of errors", which distributes the errors calculated from the loss function backwards into the network and updates the parameters using gradient descent. The gradient descent algorithm can be expressed by:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla_{\theta} C(\boldsymbol{\theta}), \tag{2.21}$$

where α is the learning rate and $C(\boldsymbol{\theta})$ is the loss function. To speed up learning speed, an optional algorithm called Stochastic Gradient Descent (SGD) updates the parameters based on evaluation of the loss function and the gradients on randomly selected subsets of the training data. Hence, SGD can be regarded as a stochastic approximation of gradient descent optimization. Besides, SGD in itself adds a regularization effort due to the stochastic nature of the algorithm. We shall cover the topic of regularization together with convolution neural networks in the section after the next.

2.4.4 Deep Neural Networks

The *universal approximation theorem* of feed-forward neural networks states that every continuous function on a compact subset R^n can be arbitrary well approximated by a feed-forward neural network with one hidden layer, a finite number of neurons, and some mild assumptions about the activation function [22]. Intuitively, this can be explained simply by understanding that combination of activation functions, where height and shift are determined by the weight and bias respectfully, can fit any continuous function [21]. This continuous function will in practice corresponds to the relationship between the input data and the ground truth which the neural network is to predict. For that reason, artificial neural networks are also called universal function approximators. On the contrary, depending on the complexity of this relationship, the number of neurons in the hidden layer may become in-feasible large. Therefore, another approach is to use multiple hidden layers with a limited number of neurons in each layer. Such network architectures are called Deep Neural Network (DNN), and is very powerful and efficient. This is explained by the fact that the number of linear regions grows exponentially with depth L and polynomial with the number of neurons n per hidden layer [23]. Therefore, the number of linear regions grows much faster for deep architectures compared to shallow architectures with nL hidden neurons.

2.4.5 Convolutional Neural Networks

Convolutional Neural Network is a variant of deep neural networks that takes images as input and constrains the architecture more sensibly by implementing three different kinds of layers: convolutional layers, pooling layers, and fully connected layers. These layers are stacked to form a full CNN architecture.

Convolutional layer

The convolutional layer is the core building block of the CNN and has inspiration from biology, where the visual cortex of organisms contains receptive fields that fire based on certain types of stimuli. Convolutional layers persuade this where learned filters are systematically applied to the input to create feature maps that summarize the presence of features in the input. A convolutional layer if visualized in Figure 2.7 along with an activation function.

A filter is arranged in three dimensions: spatial (width, height) and depth. During a forward-pass, the filter slides horizontally and vertically across the previous layer, and at each sliding step, it computes the dot product between its weights and the current region of neurons denoted as receptive fields. Then element-vice activation is performed on the output of the filter operations, which leaves the resulting spatial and depth dimensions from the filter operation unchanged while still allowing for non-linear transforms and thresholding. The resulting value is the output of the corresponding neuron in the convolutional layer. If the filter weights match sufficiently with a current receptive, then the corresponding neuron fire, analogous to the receptive fields found in biology. This

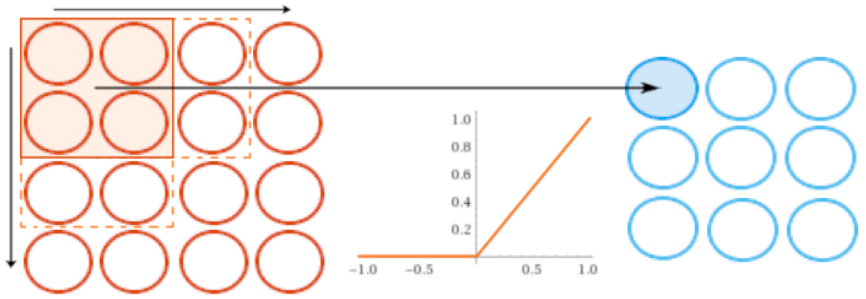


Figure 2.7: Illustration of a convolutional layer. A filter of size 2x2 slides horizontally and vertically with stride equal to 2 across the first layer. Element-wise activation with ReLU maps the dot product between the filter and the receptive field.

arrangement avoids the impractical connection of every neuron to all neurons in a subsequent layer when dealing with high-dimensional input such as colour images, which does not scale well and could easily lead to overfitting. Besides, the use of filters reduces the number of learnable weights since each filter only has a specific and low limited number of weights. The convolutional layer may implement multiple filters, which increases the depth of the layer. Note, the depth of the layer is different from the depth of each filter and the depth of the deep neural network. In addition to the spatial and depth dimension, it is common to implement what is called stride. When the filter slides over the previous layer, it can skip a certain number of neurons corresponding to the stride, which further reduces the computational cost. Zero-padding is also used to make the dimensions match.

Pooling layer

The pooling layer implements a down-sampling operation, meant to reduce the spatial dimensions, and is based on that the output from the convolutional layer may store redundant information since the filter sizes are small compared to the input size and that the receptive fields may overlap. It can be considered as a special type of filtering where one value is extracted out of multiple values based on a pre-defined rule, but the stride is always equal to the filter size. Pooling results in a reduction in the number of values and parameters needed in the next convolutional layer. Basic spatial pooling types are maximum, summation, and average pooling. A maximum pooling layer is visualized in Figure 2.8. The drawback is that spatial information may get lost due to the reduction of information, dependent on the pooling functions. The pooling layer does not have learnable parameters and works as fixed functions.

Fully connected layer

After a stack of pairs of convolutional layer and pooling layer, the extracted features are flattened into a one-dimensional matrix which by now holds information that is vital to the input. Several fully connected layers that constitute a fully connected network then learn to

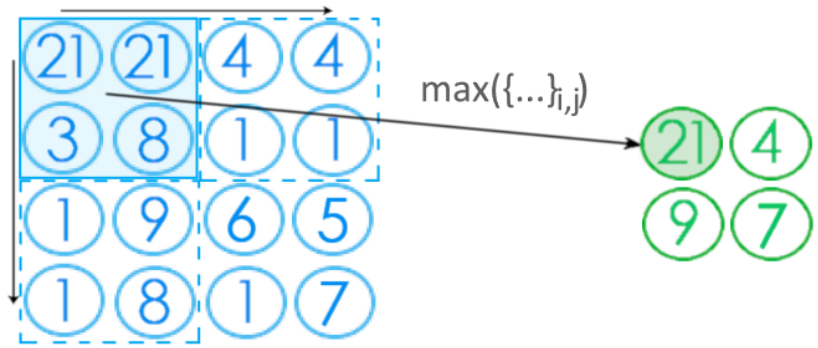


Figure 2.8: Illustration of maximum pool layer. A filter size of 2x2 and a corresponding stride of 2 is used to down-sample the output from the convolutional layer.

combine these extracted features for class prediction. The final neurons at the output layer contain an activation function dedicated to the class prediction task at hand. For instance, if the class prediction task is a multi-class one, which means that each neuron out class probabilities, it is desired that the output of the neuron is a probability distribution. The softmax activation function achieves this and is given by:

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \qquad j = 1, \ldots, K, \tag{2.22}$$

where $\boldsymbol{z}$ is the output vector before activation, z_j is the current output neuron and K is the total number of output neurons. This function sums to one over j, which is sensible in multi-class prediction since if some neuron outputs increases, others must decrease.

However, in the context of image segmentation where the output is a single-channel segmentation map, the last layers may be kept as convolutional, which preserves the spatial dimensions and spatial dependencies. The last layer then uses a thresholding activation function such as the sigmoid function (see Section 2.4.3, Figure 2.6) since the predictions are decoupled.

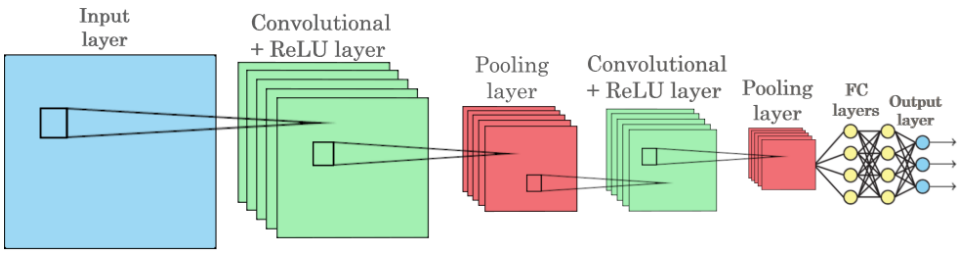


Figure 2.9: Illustration of a typical CNN architecture composed of the basic types of layers. A ReLU activation function is used, which is a common activation function in CNNs

2.4.6 Overfitting, regularization and data augmentation

Overfitting has briefly been mentioned earlier as the case when a model too closely fit a limited set of data points which may result in a failure to predict future data points. When a model contains more learnable or free parameters that can be justified by the data, it becomes prone to overfitting as large numbers of trainable parameters can describe a strikingly wide range of phenomena [24]. State-of-the-art deep neural architectures may contain millions of parameters, and efforts must be made to reduce the chance of overfitting.

Regularization refers to methods that are used to reduce overfitting by adding constraints a machine learner as to constrain the learnable parameters or information to expand the learning requirements. We have already touched upon such methods. The softmax activation function, which sums up to one forces the network to limit the prediction probabilities regardless of the number of output neurons. CNNs restricts the number of weights through filters and shared weights, and pooling, and is capable of generalizing well to avoid overfitting. Stochastic gradient descent is also in-itself a regularization effort due to its stochastic nature. There are numerous other methods, but we shall restrict us to mainly four different methods that have been used directly in this thesis, besides the intrinsic regularization in CNNs.

First, the model capacity is critical to consider to avoid having too many learnable parameters. Convolutional neural architectures often have what is called a backbone, that is a feature map extraction part in which the rest of the architecture is based upon. Choosing a backbone with a suitable model capacity for the prediction task is the first step towards avoiding overfitting. Second, limited training data is a source of overfitting because the domain that data is sampled from may not be adequately represented. Data augmentation attempts to overcome these implications, in which image transformation techniques are used to increase the diversity and to squeeze out as much information contained in the available training-set. Geometric transformation such as affine and perspective transformation, translation, and rotation are examples of such techniques. Third, data augmentation can also involve adding occlusion the images to make it more challenging and to force the neural network to increase versatility and cooperation between neurons. Randomly adding noise and black, white, or grainy squares and dots are examples of such techniques. Fourth and last, early stopping may be considered essential for avoiding overfitting. During the training of a machine learner, the validation error or score is calculated concurrently with training error or score. Training is stopped if validation error stagnates for too long or starts to increase, as this is a sign of overfitting taking place, especially if the training error continues to improve. *Early stopping* is thus to stop training at the smallest error with respect to the validation set.

2.4.7 Residual Networks

Deep neural networks achieve high accuracy by utilizing a stack of many layers (see Section 2.4.3). However, as shown by He and Sun at Microsoft Research, that adding too much depth can actually decrease performance [25]. Stacking layers can accumulate derivatives

during backpropagation, which results in the weight becoming more and more sensitive to changes the deeper one moves into the network. Regularization, correct weight initialization, and good choices of activation functions can restrict the problem to a certain degree, but eventually, the depth becomes a problem. Therefore, He and Sun et al. (2015) purposed Deep residual networks with residual blocks as a way to circumvent this problem. The residual block adds a feed-forward connection across two layers, which allows for the passing of useful concepts learned earlier in the network, as illustrated in Figure 2.10, where the identity x can pass the two weighted layer without being scrambled. The team found that residual block made networks easier to optimize and gained accuracy from considerably increased depth. The accuracy measured by mAP reached the top at around 100 layers, but decreased when going towards 1000 layers. The neural network architecture used in the experiment in Section 4 used a backbone based on Resnet101, that is a residual network of 101 layers.

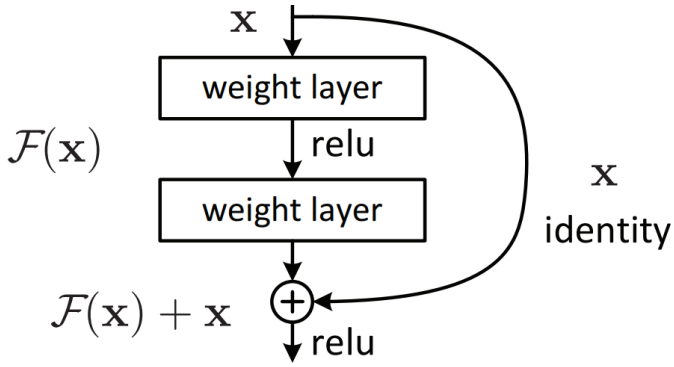


Figure 2.10: Residual learning: a building block in deep residual learning frameworks. Source: [26]

2.4.8 Mask R-CNN

The previous sections have laid the foundation for understanding how object detection and instance segmentation can be solved using deep learning. We shall now outline a convolutional neural architecture named Mask R-CNN that solves the task of object detection and instance segmentation, and which our experiment in Section 4 utilizes. Mask R-CNN is based on multiple predecessors, and to fully understand the inner workings of Mask R-CNN, we start by giving an overview. The predecessors and Mask R-CNN architectures are illustrated in Figure 2.11.

R-CNN

R-CNN or Regions with CNN features in an object detection algorithm that was developed by Ross Girshick and a team of researchers at EECS Berkeley in 2014. It relied on a Selective Search Algorithm to generate region proposals, improvement of a well-known

CNN architecture named AlexNet [27] to extract features from the proposed regions, and a Support Vector Machine to detect and classify the presence of objects based on the extracted features in parallel with a bounding box regressor for adjusting the region proposals afterwards. Selective Search is a region proposal algorithm based on creating hierarchical groupings regions based on colour, texture, size, and shape similarity and compatibility. Furthermore, Support Vector Machine is an unsupervised clustering algorithm that sorts data with the margins between clusters as far apart as possible. After years of stagnation in object detection performance, R-CNN offered a scalable detection algorithm that improved mean average precision by more than 30 % relative to the previous best results at VOC 2007, achieving a mAP of 53.3 % at VOC 2012. While these algorithms have an okay runtime, the running of the CNN architecture on each of the proposed regions massively slows down the runtime to 49 seconds per test image. Moreover, since the Selective Search algorithm is a fixed algorithm without room for improvement, it could lead to the generation of lousy candidate region proposals, such that R-CNN consequently would be wasting CNN detection which further contributed to a slowdown in speed.

Fast R-CNN

In 2015, Ross Girshick, which was now working at Microsoft Research, expanded on the idea of R-CNN. Instead of feeding all regions proposals to a CNN, the input image was fed into a CNN directly to generate what has later been named a feature map. Region proposals from the selective search that were used to pool out the Regions of Interest (RoIs) using what has been denoted as *RoIPool*, before these regions were feed into a fully connected network terminated with a softmax classifier and a bounding box regressor to adjust the region of interest. This improved runtime per image significantly to 2.3 seconds per image.

Faster R-CNN

Thinking that speed could be further improved by replacing the Selective Search algorithm, a Microsoft Research team in China proposed in 2016 instead used a Region Proposal Network for obtaining Region of Interests based on the backbone feature extraction network.

Regions were generated in the Region Proposal Network using a sliding window over the feature map with *anchors* in different sizes and ratios, and at the same time given a confidence score based on how well they lined up with potential objects. Sliding window is a traditional technique which is known to be slow, but selection based on the confidence score of the proposed regions allowed for a dramatic reduction in the number of ROIs, effectively separating background and objects. The process of selection proposals with a confidence score higher than a specific limit is denoted as *non-max suppression* (NMS). This improved runtime per image yet more to 0.2 seconds per image.

Mask R-CNN

Then, in 2017, a team at Facebook AI Research (FAIR) come up with the idea to extend the prediction tasks to more than just classification and bounding box refinement. A mask header was added as a third header for prediction object masks in parallel with the existing branch for bounding box recognition. The header is a small FCN applied to each RoI, predicting a segmentation mask in a pixel-to-pixel manner. Since Faster R-CNN's RoIPool was not designed for pixel-to-pixel alignment and led to spatial quantization for feature extraction, the team proposed a simple, quantization-free layer called *RoIAling* improving mask accuracy to as much as 50 %. Besides, it was found essential to decouple mask and class predictions by predicting a binary mask for each class independently to avoid competition among classes, in contrast to the usual per-pixel multi-class categorization in FCNs.

The final architecture involved five different dynamic parts being the backbone feature extractor, the Region Proposal Network, the RoIAlign, the bounding box recognition header, and the mask header. The backbone feature extractor can be any FCN, but the experiment of Section 4 have used Resnet101 along with a FPN as proposed by the authors of Mask R-CNN[17].

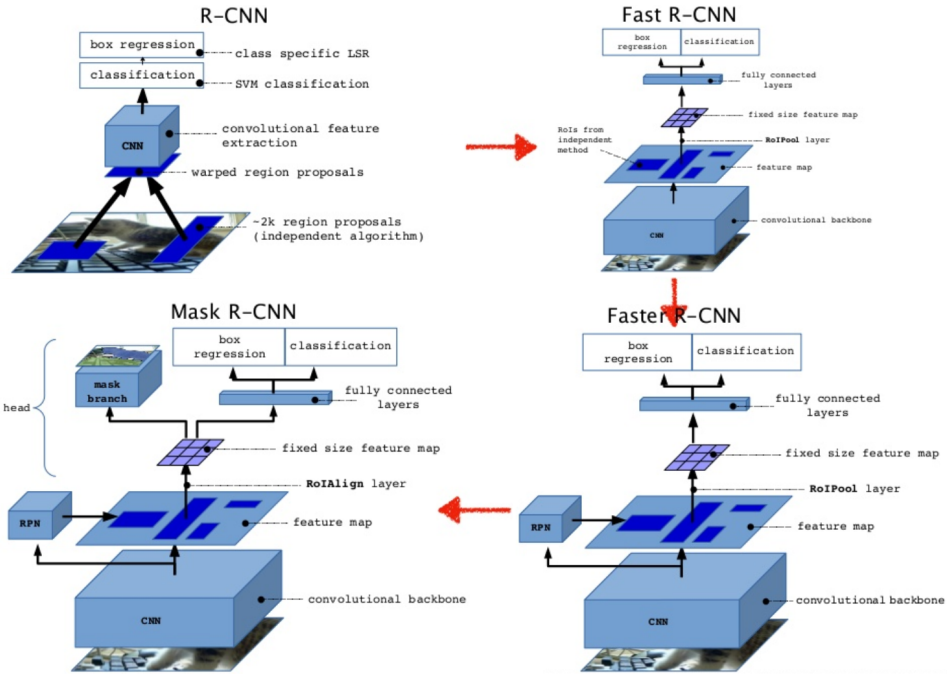


Figure 2.11: Illustration of architectures that have laid the road toward Mask R-CNN . Source: [28]

2.4.9 Feature Pyramid Network

Detecting and segmenting object in different scales can be challenging, in particular for small objects. A regular CNN progressively reduces the spatial size of the feature maps, which hurts the ability to detect and segment small objects. In 2017, a team based on cooperation between Facebook AI Research (FAIR) and Cornell University and Cornell Tech came up with a feature extractor named Feature Pyramid Network (FPN) designed based on a pyramid concept with accuracy and speed in mind [29]. It generates multi-scale feature maps, based on combining low-resolution semantically strong features with high-resolution semantically weak features via a top-down pathway and lateral connections. This architecture can better represent information from different scales in the input image. The bottom-up pathway is a regular CNN but defined in a pyramid fashion with multiple steps. The steps are connected to the top-down pathway but are first convoluted using a filter of size 1x1 for obtaining a weighted sum of spatial information contained in the depth, and by such reducing the dimensions along the depth. Furthermore, the top-down pathway is up-sampled by a factor of two using nearest neighbour in a similar but opposite fashion as the down-sampling during pooling operations, before being merged by element-wise addition in the lateral connection with the feature map from the bottom-up pathway of the same spatial size.

The team went further and embedded a FPN into the region proposal network of Faster R-CNN, persistently improving its performance. Since Mask R-CNN allows for any back-bone, it has also embedded a FPN into its resnet-backbone for creating segmentation proposals, as illustrated in Figure 2.12

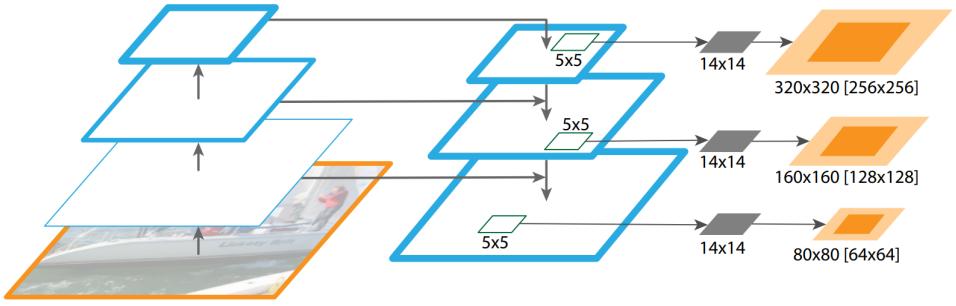


Figure 2.12: Illustration of a FPN extended for segmentation proposals. A bottom-up pathway is combined with a subsequent top-down pathway through lateral connection as to produce multi-scale feature maps. Segmentation proposals is generated by sliding anchors across the multi-scale feature maps. Source: [29]

2.5 Domain adaptation in CNN

When the domain and task changes, it can be expensive or impossible to recollect the needed training data and rebuild models, especially within deep learning and CNNs. Instead, an already trained model on a large dataset from one domain can be transferred

to another domain using a smaller dataset. This learning framework is named Transfer Learning (TL) and may be summarized as a method for improving the performance of a machine learner in one domain using knowledge learned from other domains. Domain Adaption (DA) is a special case within transfer learning, where domains are different, but labels are the same. Transfer learning does not necessarily demand sophisticated methods, and standard practice based on fine-tuning can be sufficient to obtain good performance.

The following two sections present some definitions and notations regarding transfer learning and domain adaptation and cover two different domain adaptation learning methods — the experiment of Section 4 relay heavily on the definitions and content of this section.

2.5.1 Domain adaptation and covariate shift

The section is primarily based on [30] and has inspiration from [31]. Notation has been modified slightly.

A domain D can be said to consist of two components: a feature space $\mathcal{X}$ and a marginal probability distribution $P(X)$, where $X = \{x_1, \ldots, x_n\} \in \mathcal{X}$. In general, if two domains are different, then they may have different feature spaces or different marginal probability distributions. Given a specific domain $\mathcal{D} = \{\mathcal{X}, P(X)\}$, a task $\mathcal{T}$ consist of two components: a label space $\mathcal{Y}$ and a conditional probability distribution $P(Y|X)$, where $Y = \{y_1, \ldots, y_n\} \in \mathcal{Y}$. The task can be denoted as $\mathcal{T} = \{\mathcal{Y}, P(Y|X)\}$, and cannot be observed but learned by any machine learner as a model $\widehat{P}(X|Y)$ from training data. If the training data consist of pairs $\{x_i, y_i\}$, where $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$, the learning of the model is called supervised learning. In the context of this thesis, the machine learner will be an CNN architecture. The authors of [30] considers two domains in regards to transfer learning: the source domain $\mathcal{D}_S$ and the target domain $\mathcal{D}_T$, and defines transfer learning as:

"Given a source domain $\mathcal{D}_S$ and learning task $\mathcal{T}_S$, and a target domain $\mathcal{D}_T$ and learning task $\mathcal{T}_T$, transfer learning aims to help improve the learning of the target predictive function $f(\cdot)$ (i.e $\widehat{P}(X|Y)$) in $\mathcal{D}_T$ using the knowledge in $\mathcal{D}_S$ and $\mathcal{T}_S$, where $\mathcal{D}_S \neq \mathcal{D}_T$, or $\mathcal{T}_S \neq \mathcal{T}_T$."

This definition gives rise to four different learning scenarios:

1. $\mathcal{D}_S = \mathcal{D}_T$ and $\mathcal{T}_S = \mathcal{T}_T$
2. $\mathcal{D}_S = \mathcal{D}_T$ and $\mathcal{T}_S \neq \mathcal{T}_T$
3. $\mathcal{D}_S \neq \mathcal{D}_T$ and $\mathcal{T}_S \neq \mathcal{T}_T$
4. $\mathcal{D}_S \neq \mathcal{D}_T$ and $\mathcal{T}_S = \mathcal{T}_T$

Since a domain is a pair $\mathcal{D} = \{\mathcal{X}, P(X)\}$, the condition $\mathcal{D}_S \neq \mathcal{D}_T$ implies that either $\mathcal{X}_S \neq \mathcal{X}_T$ or $P_S(X) \neq P_T(X)$. Similarly, since a task is defined as a pair $\mathcal{T} = \{\mathcal{Y}, P(Y|X)\}$, the condition $\mathcal{T}_S \neq \mathcal{T}_T$ implies that either $\mathcal{Y}_S \neq \mathcal{Y}_T$ or $P\left(Y_S|X_S\right) \neq P\left(Y_T|X_T\right)$. This gives rise to various additional learning settings.

When the target and source domains are the same, i.e. $\mathcal{D}_S = \mathcal{D}_T$ and their learning tasks are the same, i.e. $\mathcal{T}_S = \mathcal{T}_T$, as in learning scenario (1), the learning problem becomes a traditional machine learning problem. Domain adaptation, on the other hand, is a special problem that finds itself between learning scenarios (3) and (4). The domains are different due to a difference in the feature space, i.e. $\mathcal{X}_S \neq \mathcal{X}_T$, the marginal probability distributions of the input data are different, i.e. $P(X_S) \neq P(X_T)$, and the tasks are different due to a difference in the conditional probabilities, i.e. $P(Y_S|X_S) \neq P(Y_T|X_T)$, but the label space are the same, i.e. $Y_S = Y_T$. These are similar to the requirements for a covariate shift, where the marginal probability distribution of the input data changes. Domain adaptation covers the problem of training a model in a source domain, and deploying it across one or more different target domains which are separated from the source domain by a covariate shifts. As a final note, we mention the *sampling selection bias*, which denotes the case when the feature space in two domains are the same, while the selection of samples from the two domains introduces a feature distribution bias towards the domains and between the samples.

2.5.2 Fine-tuning and Gradually Fine-tuning

Modern deep neural networks are found to exhibit a common phenomenon reliably; when trained on images, they all tend to learn first-layer features that resemble either Gabor filter or colour blobs [32]. Gabon filters are linear filters and are widely used within computer vision due to their ability for extracting texture feature. In general, features in neural networks have been found to reliably transition from general in the first layers to specific by the last layers of a network. This observation has given rise to the notion about general and specific layers, based on how well or how short features at a specific layer transfers from one task to another, respectively. Furthermore, training a deep neural network from the bottom in a new domain can be expensive or impossible, as mentioned in the intro of this section.

Therefore, a common approach within deep learning is first to train a base or source network in one domain on a large dataset, and then fine-tune an arbitrary number of layers or replaced layers in another domain and possibly with a different prediction task. Layers that should be kept as they were are locked in a process called *freezing*. If the label space or prediction task changes in some way from the source to the target, it is necessary to replace the top layers of a source network with a new building block learner consisting of custom layers or other custom machine learners dedicated for the new target task.

"Off-the-shelf" methods denote such approaches since one or more shelves are taken off and replaced. The remaining layers from the source network can then either be frozen or further fine-tuned on the target domain and task along with the training of the new building block learner. The remaining layers will then, to a great extent, work as general feature extractors considering their generality, while the new building block will learn the conditional probabilities for the new target task. A study from 2016 conducted by a team from Berkeley at University of California found that the best practice is fine-tuning by keeping as many layers as possible from the source network and adjust the level of fine-tuning based on the visual distance from the source and the amount of available target

training data [33].

In domain adaptation where the label space remains the same, fine-tuning without removing and replacing layers may be sufficient. Two popular fine-tuning approaches where all layers from the source network are copied are:

- Full fine-tuning: All layers in the network are kept open during fine-tuning.
- Gradual fine-tuning: Deepest layers progressively unfroze during fine-tuning.

The first of these two approaches allows for flexibility during training by allowing all weight to be fine-tuned. The team at Berkeley found that that the more data that are used, less freezing is better [33]. Besides, the more the source and the target domains are different, the more adaptation would be necessary. Restricting the network through excessive freezing could lead to the fine-tuned network under-performing compared to it's potential. However, if the dataset is small relative to the number of parameters, full fine-tuning may result in overfitting, where the network starts to memorize the training images. Then the second of these two approaches may be more suitable since it allows for keeping general features untouched for a more extended period during the fine-tuning process. Besides, there is a tendency within neural networks to forget previously learned information when learning new information, known as catastrophic interference. Gradual fine-tuning can intuitively be though to avoid this to a more significant extent than full fine-tuning. We will, throughout the thesis, denote models that have been adapted using one of these two methods as "fully fine-tuned" and "gradually fine-tuned" models, respectively.

2.6 Synthetic data

The problem in this thesis is concerned about synthetic data in the maritime domain in the context of computer vision. Synthetic data may be defined as any information that has been artificially manufactured to imitate the real-world to meet specific needs not found or accessible in the real world. The various statistics of the synthetic data must, to a sufficient degree, match those that are expected to be found in the real world, such that the data can be used as a substitute. The motivations for using synthetic data can be summarized as overcoming constraints regarding data acquisition in the real world. For instance, this can be different constraints from costs, scenarios, sensitivity or regulations of data, and labelling accuracy.

Convolutional neural networks have within computer vision shown outstanding performance and the ability to resist overfitting but pay in terms of requiring a massive amount of data during training. The acquisition process of labelled data can be laborious, costly, and in some cases, restricted due to real-world constraints on scenarios and regulation of data. Synthetic data can ease this process as it comes in vast quantities offered by automatic generation and high-quality labelling with pixel-vice accuracy. A German team at the University of Darmstadt injected a wrapper between a game running on a computer and the operating system, allowing them to record, modify, and reproduce the rendering pipeline [34]. After having implemented the tool, they were able to create pixel-level se-

mantic segmentation for 25 thousand images within 49 hours, an astonishing volume considering the labelling time and pixel-level accuracy. In contrast, the CityScapes Dataset contains high-quality pixel-wise instance-level annotations for 30 classes and 5 thousand images [35] and required 1.5 hours labelling per image. Besides, high-quality labelling contributes to increasing the value of the data. A paper published by a team at Massachusetts Institute of Technology (MIT) showed that a larger coarsely-annotated dataset could yield the same performance as a smaller finely-annotated one within semantic segmentation [36]. Furthermore, automatic labelling can, in addition to class labels, also offer other labels as well, i.e. depth, mesh, hierarchy, speed, and heading, which may require an immense manual workload.

Today's computer games and simulators offer highly realistic graphics to the human eye, enabled by computer graphics and 3D models that imitate light, shapes, and combinations of shapes. Furthermore, mixing of synthetic and real-world data, combinations of real-world objects and data, or other variants can also be considered synthetic data and help increase the volume and diversity of datasets. However, synthetic data also comes with a bias as they introduce features not present in real-world digital images. This bias can contribute towards making models trained on synthetic data under-perform in the real world.

Chapter 3

Computer Vision: Explainable AI

This chapter takes a look at a new paradigm within artificial intelligence, namely Explainable AI (XAI). We start by giving a general presentation regarding the rising concern about obtaining explanations within artificial intelligence, before covering a method named Local Interpretable Model-agnostic Explanations (LIME) and proposing a way to extend it to instance segmentation.

3.1 How can you trust your AI?

The success of AI can be explained by the inherent capability in machine learners to acquire knowledge and to successfully act with respect to a given narrow objective based on pre-programmed algorithms. While the algorithms become more sophisticated, our human ability to interpret internal workings decreases, and they perform as an algorithmic "black box". The community, academia, authorities, and the industry sees a growing need for ensuring that the much-needed beneficial AI systems are trustworthy in that they behave accordingly; in that they can comply to societal, moral, and legal norms [37] if desired, in that they make rational decisions and recommendations, in that they satisfy performance and robustness requirements, and in that they do so in all future time. Explainable AI is the desire to enable machine learners to provide a rationale of how they arrived at their decisions or recommendation, to characterize their strength and weaknesses, and to do so in ways that are interpretable to humans [38, 39]. Such machine learners may not only meet legal regulations and become trustworthy for future use, but may also help to reveal flaws, bottlenecks, and provide suggestions for improvement in the model design.

The first question is about the requirement for the explanations beyond the requirement of interpretability. Firstly, it is essential that the explanations are honest, such that there is a consistency between the internal decision-making process of the machine learner and its explanations. This honesty will both ensure that human can trust the explanations,

and force the machine learners to act according to the given explanations. Secondly, the content of the explanations must be representative of the workings of the machine learner, such that factors important for the internal decision-making process are also included in the explanation. Thus, telling the truth means telling the whole truth. Lastly, the size of the explanations must conform to the complexity of the machine learners, proportionally speaking. Naturally, complex models have complex internal workings, and therefore, the explanations should also be expected to be comprehensive. While complexity is seen as the source to the black box nature om machine learners, Montavon et al. (2017) [40] argues that recent advances in interpreting neural networks allow users deeper insights unavailable from simple models because of their complexity. This claim makes sense, considering their assumed ability to provide meaningfully complex explanations which simple models cannot provide.

The second question is, what requirement should be placed on the content of the explanations for them to satisfy the demand for explanations. The demand is highly dependant on multiple factors, e.g. the purpose and workings of the machine learner, the purpose of the explanations, and for whom the explanation is meant. For instance, the EU's General Data Protection Regulation (GDPR) imposes a legal right for explanation to the data subjects, being the natural person to whom data relates, but is vague in its description for what content in the explanation is required [41]. In the context of instance segmentation and a desire to obtain trust in a model, a technical explanation based on saliency maps and information flow charts in the neural network would perhaps be an overload considering the purpose of the explanation. Instead, an explanation based on the unique characteristics of an object in an image would perhaps be more suitable. Such an explanation could contain related characteristics to the object, but also discriminating characteristics between the object and other similar but different objects. Hendricks et al. (2016) [42] proposed a neural network with embedded explainability that was able to generate explanations that were both image relevant and class relevant. They were able to both classify the bird in each image, give a description of the bird in the image, a definition for the predicted class, and a visual description, as seen in the results in Figure 3.1. A good match between the description and the definition, along with a meaningful visual explanation, asserted trust in that the model successfully considered important and unique characteristics of the bird. In addition, a false-positive classification would also explain itself by implicitly telling what the basis for the false-positive classification was.

The bird classification application is an example of *explainability by design*, one of three common approaches for obtaining explanations. It introduces the third question; how should explainability be incorporated into an AI application? Several approaches for obtaining explanations exist, and can roughly be divided into model-agnostics approaches, algorithm-specific approaches, and explainability by design approaches. Model-agnostic approaches are black-box explainers, which is based on perturbing the features in the input and observing the output, which gives local and features specific approximation of the model's response. Such approaches wrap another component around the machine learners, and will often approximate a complex model with a simpler one that is interpretable to humans. Algorithm specific approaches inspect the inner workings of a model, such as decision tree splitting conditions, neural network information flow, or saliency maps. Explainability by design aims closest towards the true goal of XAI, that is enabling the

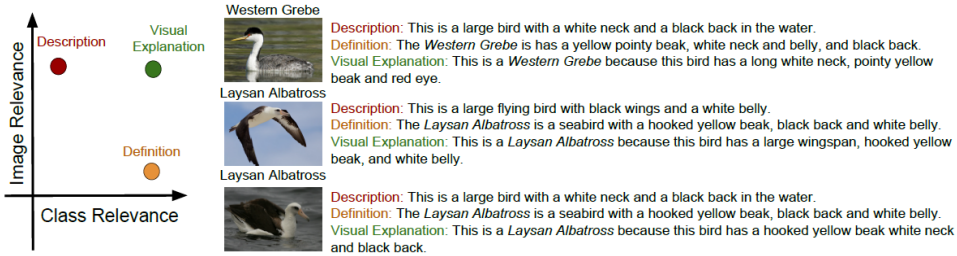


Figure 3.1: Visual explanation for bird classification based on class and image relevant description. A neural network generated explanations that were both image relevant and class relevant. In contrast, descriptions are image relevant, but not necessarily class relevant, and definitions are class relevant but not necessarily image relevant. Source: [42]

machine learners to offer explanations by them self. It is important that explainability does not harm the performance of the machine learners through a trade-off. Instead, it is desired that explainability achieves synergy with the machine learners, and helps improve performance and model design as mentioned earlier. Model-agnostic approaches are in principle applicable to any machine learner, does only reduce runtime, but are limited in their capability to explain complex models due to their simpler approximation. Algorithm specific approaches are model specific and may be complex, but allows for inner insight to the workings of the model. Comparably, these approaches require more expertise and effort than model-agnostic approaches. Explainability by design are model specific and requires expertise to be implemented. In addition, they will also be part of the development process and adds complexity to the model. They are most prone to affecting the performance, but the explanations can be expected to be dedicated to the purpose of the machine learner and hence yield the best explanations.

3.2 Local Interpretable Model-Agnostic Explanations

In 2016, Riberio, Singh and Guestrin released a paper named "Why Should I Trust You: Explaining the Predictions of Any Classifier" [43]. This section is primarily based on this paper and uses the same terminology and definitions. Within this paper, the authors propose a method named Local Interpretable Model-agnostic Explanations (LIME) as a means of "providing explanations for individual predictions as a solution to the 'trust the prediction problem', and selecting multiple such predictions (and explanations) as a solution to 'trusting the model' problem." They describe it as a novel explanation technique that can "explain the predictions of any classifier or regressor in a faithful way, by approximating it locally with an interpretable model." The overall goal of LIME is to identify an interpretable model over the *interpretable representation* that is *locally faithful* to the classifier or regressor. By interpretable model, it is meant that it is understandable to humans, regardless of the actual features used by the model. Furthermore, by locally faithful, it is meant that an explanation exhibits local fidelity, i.e. it must correspond to how the model behaves in the vicinity of the instance being predicted. In its essence, LIME enables the

user to perturb input to a model in different ways that makes sense and observe how the predictions may change, before it builds a few explanations on individual predictions that are presented to the user, such that they are representative of the model. The user can then evaluate whether or not to trust the model for the specific tasks associated with the inputs.

Behind the workings of LIME lies the assumption that every complex model is linear on the log scale, i.e. it is to be expected that two very similar observations should behave predictably even in a complex model, and that it is possible to approximate a simple model around a single instance that will mimic how the global model behaves at that locality [44]. Moreover, a simple model learned on the change in predictions based on perturbation of the input to an underlying complex model becomes model-agnostic. Thus, LIME is based on the intuition that it is much easier to approximate a black-box model by a simple model locally. A simple linear model is learned by fitting the simple model to the perturbed prediction instances by their similarity to the instance that is to be explained, as illustrated in Figure 3.2. Then, features are selected from the simple model and used as an explanation for the complex model behaviour. The simple model to use, the method for perturbation, the measure of similarity, and the method for feature selection are dependent on the complex model and the prediction problem.

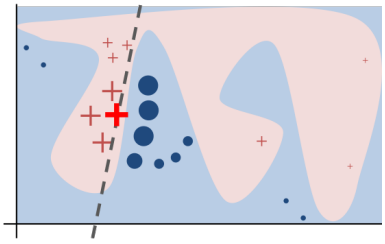


Figure 3.2: Illustration of local faithful explanation obtained from LIME. A black-box model's complex decision function f (unknown to LIME) is represented by the blue/pink background, which cannot be approximated well by a linear model. The bright, bold red cross is the instance being explained. LIME samples instances, gets predictions using f, and weighs them by the proximity to the instance being explained (represented here by size). The dashed line is the learned explanation that is locally (but not globally) faithful. Source: [43]. (*Note: the description is modified slightly)*

Explanations within image classification are illustrated in Figure 3.3, where an explanation is obtained for an image classifier based on the task of predicting how likely it is for the image to contain a frog. In this case, the image is perturbed by using different collections of super-pixels obtained by any regular image segmentation algorithm, i.e. the watershed algorithm. After having performed a certain number of detection steps, the super-pixels are weighted by regression based on the associated probability scores and K super-pixels or features are put together though feature selection, i.e based on highest weights, to produce an explanation. In this case, the head of the frog is considered an important feature for obtaining predictions with high probability and given as an explanation.

The runtime is highly dependent on the complexity of the black box. The authors of the paper mention that explaining predictions from a random forest with 1000 estimators in a laptop with 5000 steps took around 3 seconds. On the contrary, explaining each prediction from the Inception neural network architecture for image classification took around 10 minutes.

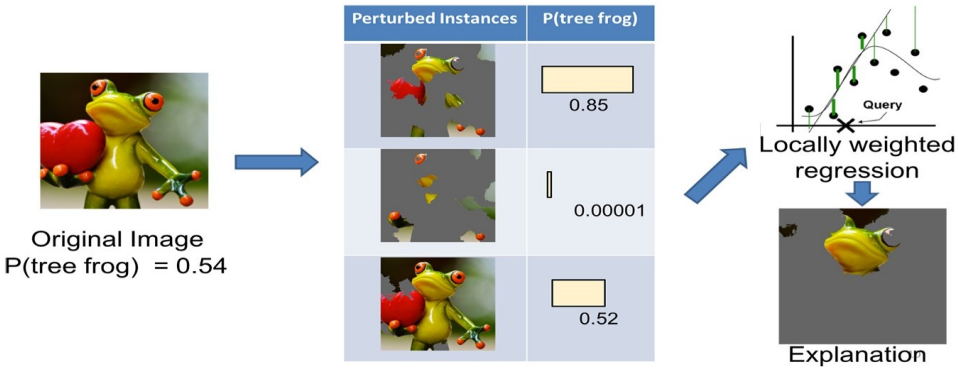


Figure 3.3: Illustration of acquiring an explaining of a prediction with LIME. Source: [45]

3.2.1 LIME for instance segmentation

As mentioned in the background of the introduction to this thesis, it would be interesting to see if recent methods within the area of XAI could be used to obtain explanations and insight into the workings of instance segmentation models. We saw an example in the previous section for how LIME could be used for obtaining an explanation for image classification, and therefore it is natural to ask our self whether it is possible to extend the method to image segmentation or precisely instance segmentation. It would be interesting to obtain an explanation not only for the classification of objects but the mask as well. In theory, since LIME is a model-agnostic method which relies on perturbation of input and the scoring of predictions, it should be possible to extend it to give explanations for masks predictions as well as the detection confidence. It is possible that the explanations for these two tasks will be different, since they may depend on different characteristics of objects.

Therefore, it is proposed to give two different scores for a single detection in an image, based on the detection confidence and the mask quality measured by comparison to the originally predicted mask. These two scores will consequently give two different explanation for each detected object in an image. This proposal is investigated further through an experiment based on a trained instance segmentation model as a part of the experiment in Chapter 5. We delay a further description of the details of this experiment. Furthermore, there is a possibility of cropping images around detected objects based on the predicted bounding box. This cropping will effectively decrease the runtime, as the number of segments could be limited while having semantically meaningful segments of objects. However, the drawback is that the relationship to surrounding environments will get lost. This possibility is investigated in the experiment, as well.

Chapter 4

Synthetic Data acquisition and labelling

The process of data acquisition, preparation, and labelling of instance-level labelled synthetic data by using Kongsberg Digital's K-Sim have been so extensive that it is justified to dedicate a separate chapter for explaining the details this process. Therefore, we will in this chapter first present K-Sim and what it can offer, before explaining in detail the delicate method develop for acquiring instance-level labelled images from the simulator based on screen captures and texture mortification. Run-times of the method are mentioned, while time consumption statistics related to labelling of synthetic data is delayed until the next chapter and compared to manual labelling of real-world data. In total, this chapter gives insight into the practical implications of using K-Sim for the generation of labelled synthetic images in the maritime domain. Later, we will in Chapter 7 as mentioned in the outline in Section 1.4 discuss the acquisition method in light of what it can offer and its limitations. This discussion serves as a basis for the related conclusion in the final and conclusive chapter of this thesis.

4.1 K-Sim

Kongsberg Digital's K-Sim simulates highly realistic maritime scenarios and provides maritime education, training and studies. It provides a wide range of vessels and object that behaves as in the real world, multiple geographical areas and all possible weather condition. These possibilities enable a vast amount of scenarios, such as navigation, ship handling & manoeuvring, towing and tugging, High Speed and RHIB, Inland Waterways, Search and Rescue (SAR), and many more.

Simulations are created before the start and later controlled during implementation through the user interface, as shown in Figure 4.1. Vessels and other models such as buoys and

ice are inserted into a simulation using a drag-and-drop fashion and behave according to predefined behaviour. As far as they are able, they can follow predefined routes as shown in Figure 4.2, receive and follow course and speed orders during the simulations, or maintain their position through dynamic positioning. Multiple users can also take direct control of the vessels at the same time, or interact with the different objects. The vessels are authentic models of real-world vessels, and therefore, the visual characteristics of these vessels can be found in the real world, making them highly realistic. The camera perspectives are also similar to that in the real world, resembling how we see objects, waves, clouds, and land environments in the real world. Light conditions are diverse and imitate those in the real world, but since the engine technology and computer capacity set some limits, it is naturally expected that a covariate shift is present. The simulator is capable of simulating any weather condition such as wind, waves, swell of sea, cloud types and density, rain, fog, hail, snow, sea colour, at any time of the day.

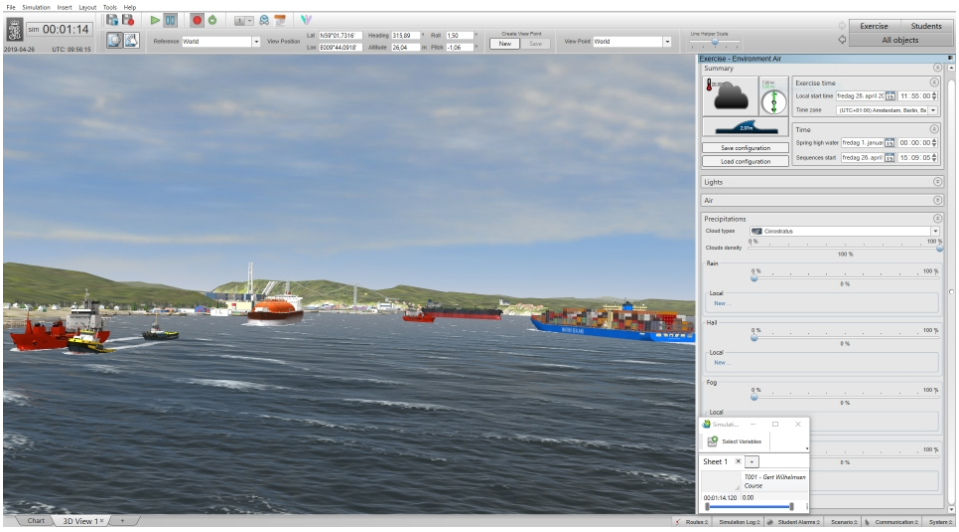


Figure 4.1: K-Sim user interface with 3D-view option. The upper panel controls the runtime of the simulator and the configuration of the camera perspective. The left side-panel controls the setup, planing of scenarios, and all the states in the simulator. All the logs in the simulator are accessed through the bottom right panel and presented in a new window frame. The open log window shows a timestamp in the right bottom corner for one of the states in the simulator. In the bottom left corner there is a menu for different views, among them being the chart and 3D-view.

During the simulation, a detailed overview of all the states in the simulator is provided to the user through the user interface at a sample rate down to a hundred-millisecond resolution. This information is accessible both through the user interface during the simulations and as downloadable .csv files with timestamps. The camera angle can mainly be configured in two ways, either in a fixed position and perspective in the world or attached to an object. However, this can be changed at any time. The simulator runs in real time, but the recording of each simulation can be saved and replayed at different speeds from 0.1 up to

10. The recordings contain all the information about the simulations, and can, therefore, show the simulations from any desired perspective.

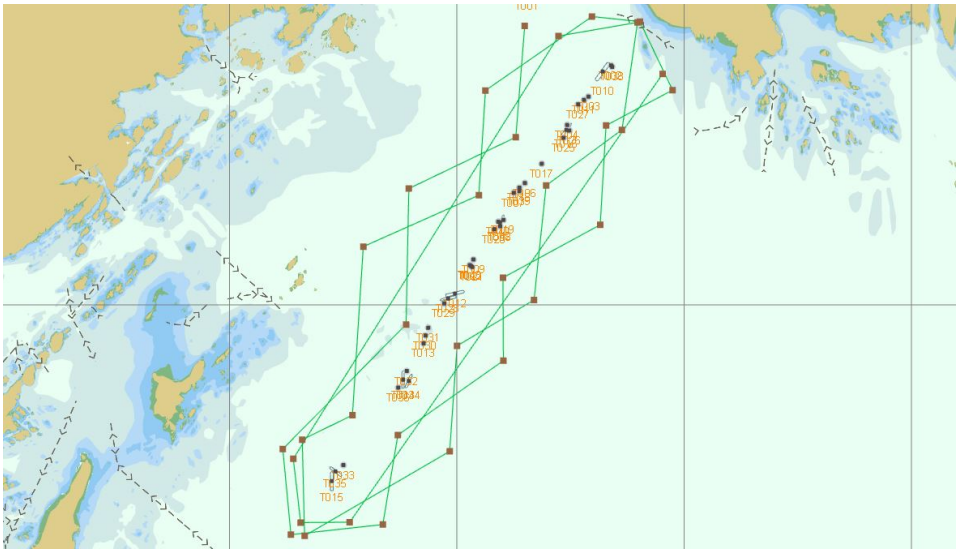


Figure 4.2: K-Sim user interface chart view. A route is shown as the sequence of connected green lines. Object are aligned en encapsulated by the route.

Before simulations can be created, the model of each object that is to be inserted into simulation has to be deployed using K-Sim's deployment tool. Each object is created by using the deployment tool to compile it's model, containing the physical characteristics and the texture with determines it's behaviour and looks, respectively. A model library was made available for this thesis, with both restricted and unrestricted models. The restricted models were allowed to be used, but not displayed or mentioned by name such that they could be recognized in the thesis. In addition, multiple geographical areas were also made available, which included cities, harbours, and regular shores. Due to the restriction, we do not specify the number of models in the model library and the numbers and names of geographical areas. There is a restriction of 100 objects per simulation, due to the increasing demand for computational power per object added to a simulation. The laptop provided by Kongsberg Digital was able to include up to approximately 50-60 objects before running into performance issues and crashes.

The simulator does not offer any option for screen recording or periodic screen-captures, nor scene segmentation information through its user interface. Moreover, no access to any application programming interface or to the internal workings of the software was available during the work with the simulator. Therefore there was no way to obtain images or scene information directly from the simulator, which made data acquisition a challenging task. However, based on a proposal from industrial co-supervisor Thorvald Grindstad, it was discovered that it was possible to change the look of objects by changing the texture in their models between simulations and replays of recordings. Besides, screen-captures

gave high-resolution images of the simulator interface and the 3D-view. These possibilities were in combination with making information about timestamps visually available in the user interface, utilized in a method developed for acquiring instance-level labelled images based on capturing the screen while running the simulator. The following section covers this method.

4.2 Data acquisition

K-Sim does not offer the opportunity of directly acquiring images together with ground-truth masks from simulations. Therefore, a method was developed for creating instance-level labelled images based on creating, carrying out, and saving the recording of a simulation, taking screen captures during replays of the recording, and exploitation of the possibility of changing textures between simulation and replays of a recording. A brief overview is shown in Figure 4.3 and in the description given below. The following section will expand upon each of the steps of the method.

The task of creating instance-level labelled images from K-sim simulations was defined as: *capture two sufficiently similar images where one image has the texture of objects exchanged by any uniformed color that is not in the blue range of the color spectrum, and use these two images to obtain masks for the other image with objects having the true texture.* This gave the opportunity of labelling objects using any arbitrary uniform color not being a part of the blue range of the color spectrum.

1. The first step of the method was about setting up a simulation in a way that dynamically produced different 3D-views when later being carried out and recorded.
2. The second step was to take synchronized pairs of screen captures from the 3D-view by replaying the recording twice, with the difference being a change in texture between the real texture and an arbitrary uniform color texture with a color that was not in the blue range of the color spectrum. The screen captures of images with objects having real texture constituted the data, while the screen captures of images with objects having uniform color was used for extracting the masks. These two types of images are further in the context of image labelling denoted as the target images and the source images, respectively. The second step also involved manual labour based on filtering out bad synchronized pairs of screen captures.
3. The third step was to extract the masks from uniformly colored objects in the image, by clustering pixel of non-background dominant colors and smoothing the resulting masks by morphological noise removal. The clustering of pixels can be divided into mainly three sequential steps:
    3.1. The first step was to threshold the image in the HSV color space to remove as much of the background as possible, based on predefined thresholding limits, before dissecting the resulting image into spatially closely connected regions.
    3.2. The second step was to cluster pixels in spatially closely connected regions based on similarity in the hue property of color and proximity in the image.

This was achieved using Bayesian Gaussian Mixture Models (BGMMs) for representing the different hue and pixel proximity distribution.

3.3. The third was to cluster the different hue and pixel proximity distributions together using hierarchical clustering of hue color centre.

3.4. The fourth and final step of the mask extraction step was to dissect the resulting clusters into spatially closely connected regions again.

4. The fourth and final step was to assess and adjust the extracted masks, based on accepting or rejecting false positive masks and vertically and horizontally adjust the accepted masks to compensate for synchronization error.

It should be mentioned that while the mask extraction may seem like a trivial task when looking at the overview in Figure 4.3, the task quickly becomes a challenge. The light condition in the simulator affects colors in a different range of the color spectrum differently in terms of change in the bandwidth, brightness, and contrast. This discrimination of colors makes it harder to cluster colors together correctly, especially when objects with close uniform colors start to overlap. Moreover, environmental noise mitigates the colors, and coastal areas and land constructions that may be present in the background can have a range of different colors. Besides, the change of texture to uniform color texture files did not always result in uniformly colored objects. This effect was due to models using multiple different texture files, which gave inconsistent variation in the color of the objects. Even worse, some of the texture files were gray-scale only. The mask extraction step had to overcome all these challenges to be reliable and to allow for flexible use of the simulator when creating labelled data.

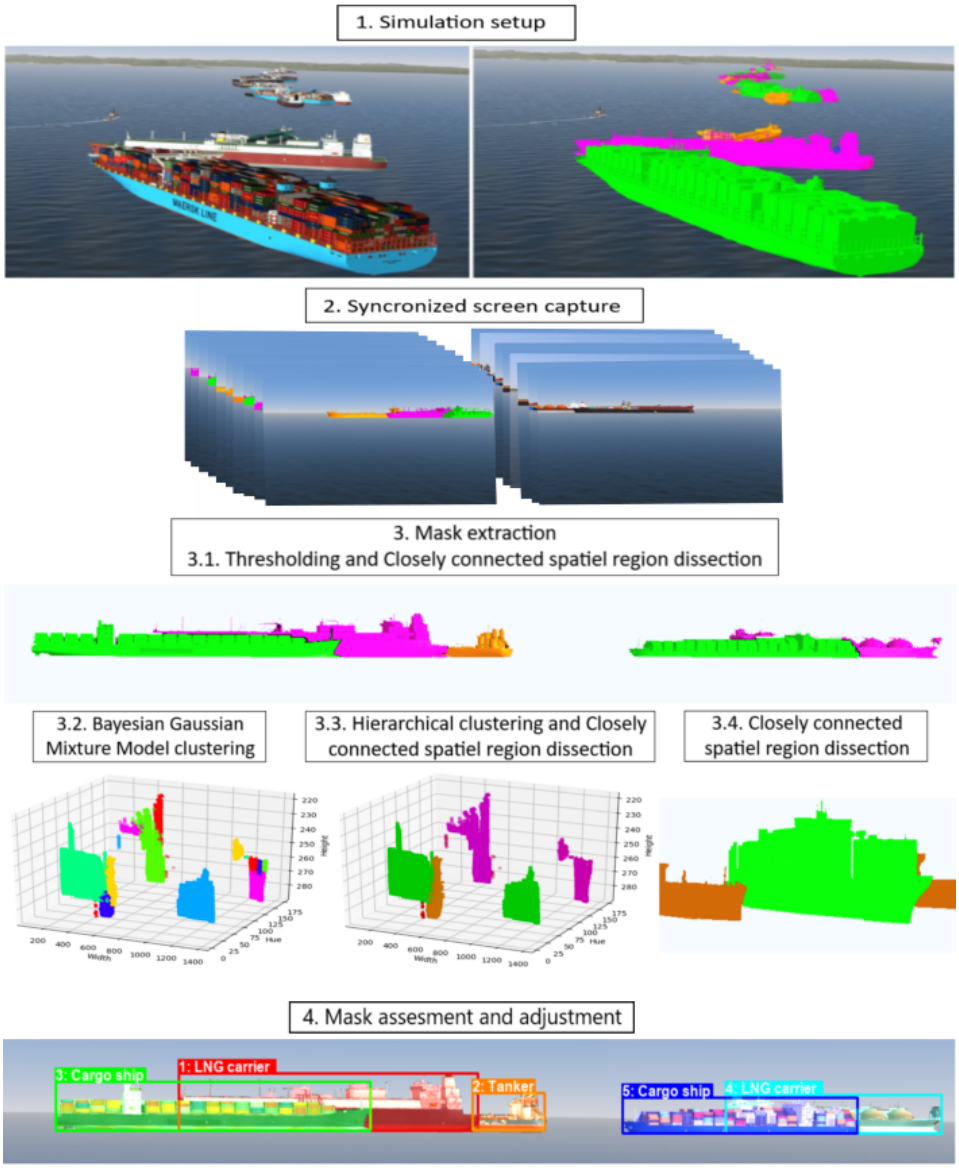


Figure 4.3: K-Sim instance-level labeling method overview

4.2.1 Simulation setup

To acquire images through the use of K-Sim, simulations had to be created manually before carrying them out and sampling images. The creation of a simulation involved mainly four main tasks:

- Collection and deployment of object models, and location of created objects in a directory where K-Sim could find them, before creating a simulation
- Insertion of objects into the simulation
- Creation of a route that a fast, agile, and wave resilient vessel with the viewpoint attached could follow to automate the change of viewpoint
- Planing the environmental noise throughout the simulation

The three-step process had to be done such that carrying out a screen capture process by running the created simulation would yield images with diverse viewpoints and objects, and within a short time period. The vessel that would be carrying the viewpoint around in the maritime virtual world and the route that it would follow are further denoted simply as the viewpoint vessel and the viewpoint route respectively. The solution to the four main tasks is covered below, while the three characteristics of the viewpoint vessel are elaborated later in the next section.

The deployment of objects first involved, as briefly mentioned in the last section, choosing a collection of object models. At this step, the different classes of objects that would appear in the simulation had to be decided upon. As we explain later in Section 4.2.2, the number of different classes of objects was depended on the encoding of object classes in a change of texture for labelling. Also, a model of a fast, agile, and wave resilient vessel that would be the viewpoint vessel had to be picked. After having gathered a collection of models, they had to be separately deployed using the deployment tool. A script was written to automate this process given that all desired models were listed together in a text file. The script would create objects by deploying the models listed and collect them together in a new directory located where K-Sim could find it. Later, this directory would be used by the K-Sim to list available objects for insertion into a simulation, if having the correct name "Object". Changing the name but keeping the directory in the same location would make K-Sim unable to locate it. This way of creating, storing, and naming object directories utilized the path variables K-Sim used to find the correct object directory and made it easy to change available objects in the simulator later on.

After having prepared the objects, a simulation would be created, and objects would be inserted into the maritime virtual world. The insertion of objects and planning of the route that the viewpoint vessel would follow was done more or less concurrently. A setup that was mainly used was to align every object after each other, and lay the viewpoint route around them. This type of setup is shown in Figure 4.2 from the last section.

The viewpoint that was attached to the viewpoint vessel would then be configured to have a fixed perspective from the viewpoint vessel throughout the whole simulation. K-Sim had the option of saving such a configuration, making it possible to reliably obtain the same dynamic 3D-view for all replays of a simulation. This option is essential for obtaining pairs of sufficiently similar target and source images, and later accurate masks.

The last step is to plan the amount of environmental noise such as fog density, and rain, hail and snow intensity throughout the simulation. This is partly for allowing the simulation to be carried out under a minimum of manual control, like controlling the weather. However, most importantly, it was to keep the mitigation of colors in the simulator more or less

constant due to the way mask would later be extracted. This is made clear in Section 4.2.3 that is about mask extraction.

4.2.2 Synchronized Screen Capture

To achieve accurate masks of objects in an image, a pair of target and source image has to be sufficiently similar. Naturally, this could be achieved by taking screen captures of the same 3D-view at the same time in two replays of the same recording, which gave rise to two different tasks:

- Obtaining the same 3D-view between the target and source image
- Replacing of real texture files with uniformly colored texture files
- Reading the simulator log timestamp

The first part of the solution to the first task has already been covered in the last section, and was based upon attaching the viewpoint to a viewpoint vessel, and then saving that specific configuration of the viewpoint. The fast and agile characteristics of the viewpoint vessel should be apparent, as to enable high speed and fast manoeuvring, which is essential for being able to change the viewpoints rapidly. The wave resilient characteristic, on the other hand, was essential for obtaining a stable 3D-viewpoint between two replays. This aspect becomes apparent in the first part of this section, which covers the main challenge and the solution for obtaining sufficient similarity between the target and source image in a pair of synchronized screen captures. The second part of this section briefly describes the method of replacing texture files having uniformly colored texture files. The third and final part of this section explaines the solution for reading the simulator log timestamp in the screen captures for enabling screen captures at the same time.

Obtaining the same 3D-view between the target and the source image

During trial and error, it was discovered that two replays of a simulation viewed from the same view-point would not necessarily create the same sequence of frames. Frames displayed at precisely the same simulation log timestamp with 100-millisecond resolution would therefore not necessarily be exactly alike. Even worse, the sample rate was only between 8-12 samples per second, and two replays of the same recording would not yield the same samples and timestamps. Therefore, it was not possible to guaranteed that the two pairs of images taken at the same time would be sufficiently similar, as to obtain accurate masks.

This issue was solved by first frequently taking screen captures and labelling them with the log timestamp, before attempting to match the timestamp in screen captures in the next replay. The timestamp of the new screen capture was compared to the old timestamp, and a pair of images was omitted if the difference between the timestamp excited a given threshold. The threshold was after trial and error set to ± 25 ms. To make this way of synchronizing screen capture efficient, the screen capture was adjusted as to contain both the

3D-view and the part where the simulator log timestamp was made visible. The timestamp would then be read, and the 3D-view cropped out of the screen capture if the difference between timestamps was within the threshold. This trick avoided a source to higher synchronization error by first having to take a screen capture of the simulator log timestamp, before taking a new screen capture of the 3D-view. There was no loss in image quality due to cropping, because it does not involve any re-size transformation of the screen capture.

After obtaining a sequence of pairs of images that all satisfied the error threshold, they were compared by visually inspecting the difference in color-space values. A script was written to iterate through the pairs of images and show their color-space difference in an ordinary 2D image. Precise pairs of images would give a difference only for the objects since they would have their texture changed, while everything else would be empty. Imprecise pairs of images would, on the other hand, be noisy, and the displacement between the objects would be visible. Two comparisons of one precise pair and one imprecise pair of target and source image are shown in Figure 4.4. A reliable filter for automatically identifying and extracting precisely synchronized pairs of images could have been implemented based on finding the amount of noise in the image. However, with proper adjustment of the masks, many of the imprecise synchronized pairs of images were still useful if the objects did not intersect with both sides of the images. The process of adjusting masks is explained in Section 4.2.4. The time it took to inspect pairs of images would vary, since the quality of synchronizations varied throughout a simulation and between simulations. This variation is most likely due to variation in hardware performance. It was measured that it took around 36 minutes to inspect 2182 images.

To limit time consummation, the speed in the replays was set to 2, effectively reducing the time it took to replay simulation and taking screen captures. Empirically shown, this improved the quality of the synchronization between the target and source images. This improvement can be due to a change in the ratio between frame rate and timestamps per second, which may give a more consistent creating of frames.

With speed set to 2, it was found that 2 out of 3 attempts of synchronization was rejected. Based on the speed of the viewpoint vessel, the period between each screen capture in the first replay was attempted to be set to a high enough value such that the pairs of images after synchronization and inspection would be distinctive as not to introduce a source to bias in the dataset by creating dependencies between images. Objects at close range would end up with being captured from 1 to 3 different angles, while objects at long range could end up with being captured from between 1 to as much as 8 times.

Replacing of real texture files with uniformly colored texture files

The texture files for a specific model are all collected in the same directory, but with possible sub-directories. The texture files are based on multiple different file-types, like regular .png -and .jpg-files, and special file-types as gray-scale .inta files. Various file-types with same colors as far as they were able, did not necessarily result in the same color in simulations. An object that did not obtain sufficiently uniform color was removed, and colors that did vary to much due to light-conditions was avoided. A color-file library was created where all colors had one of all possible file-types as far as the file-types was able. Then a

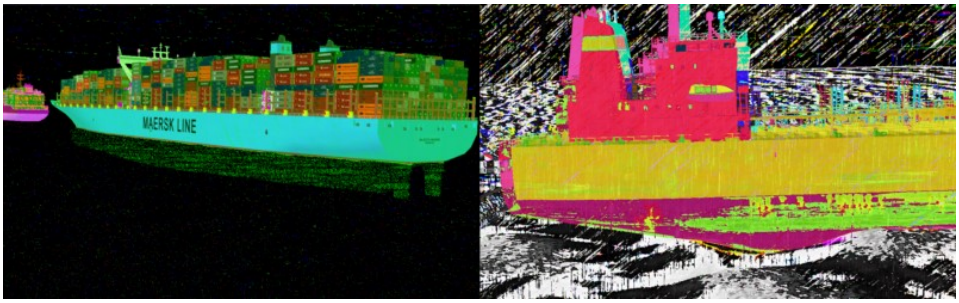


Figure 4.4: K-Sim target and source image compared by differentiating color-space. Left side shows comparison of a precise pair of images. Right side shows comparison of an imprecise pair of images.

script was written to automatically iterate over multiple models and give them specific uniform colors. It made a copy of the folder and file structure of the texture directory for each model and assigned it the same name as the original texture directory, before replacing each texture file with the corresponding uniform-color texture file.r This way, a collection of models could automatically have their texture files replaced by given uniform colors. The deployment script would then deploy all models into objects. By pairing a objects directory with original texture to its uniform color counterpart, the exchange in the look of the objects in the simulator was just about changing which object directory to use by naming it "Object". K-Sim would then find it through the path in the software.

Reading the simulator log timestamp

In order to read the timestamps, a Random Forest Classifier (see Section 2.2) provided by Scikit-learn[8] was used to build a classifier for the digital digits in the simulators log timestamp which achieved an accuracy of 100 percent on an acquired training dataset and was able to pass a simple but effective test consistently. The classified digits were then used to calculate the time. A training dataset consisting of 76 images of timestamps was first sampled with a filename corresponding to the label of last 5 digits in the timestamp. This way of manual labelling was both fast and accurate. Each of these timestamps was then dissected by cropping into 5 distinct images of digits, giving a total of 380 digits. A binary image of the digits was then created by:

1. Conversion into a gray-scale image.
2. Re-scaling into a 12x12 image by bi-cubic interpolation.
3. Gray-scale inversion.
4. Thresholding for binarization of digit and background.
5. Filtering by keeping the largest connected component with connectivity equal to 4 to remove noise.

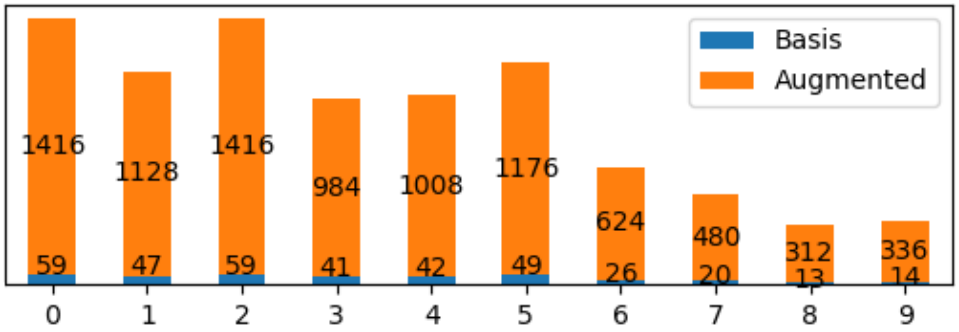


Figure 4.5: K-Sim log-clock digit training-set for building a Random Forest Classifier. The training-set is heavily unbalanced, but did not have any noticeable implications.

Each of these binary images of single digits was then augmented by translation both horizontally and vertically by 1 and 2 pixels in all directions, giving additionally 20 images per digit. Augmentation by translation was considered necessary to make the classifier robust to imprecise adjustment of the simulator log clock timestamp in the screen. Although it was considered to centre the centroid of the binary images for the same reasons for augmentation by translation, the former was empirically shown through testing to be sufficient. Centring of centroid could also have given unpredictable behaviour if some of the digits was partially cropped. Unfortunately, the training set ended up heavily unbalanced in terms of the number of digits per digit (Figure 4.5), since the sampling process inherited a distribution bias for digit appearance in simulator log clock. However, this did not become a problem as the test developed for verification and described below showed itself to be sufficient for avoiding any problems during usage of the models. Besides, Random Forest Classifier is resilient against dataset unbalance, which may explain why this did not become a problem.

A test dataset was created by collecting screen captures of log clock timestamp sequentially. In total, 100 timestamps from a period of 5 minutes were collected. A simple test was then based on classifying the time and then later comparing it to the former and later classification of timestamps. If the classification showed a non-monotonic growth in time, the model would be considered to be faulty. While the test could have been more robust by being extended to having from a few hundred milliseconds to many hours in time difference, the models that passed the test during the later experiment never failed, and thus there was no need for improvement.

A random forest classifier for the timestamp digit was then build using between 60 and 100 estimators. A dedicated and straightforward method was used to determine this hyperparameter. In order to achieve fast classification of the timestamps, it was desirable to keep the number of estimators low while still sufficiently high as to pass the test. This desideration was achieved by first building and testing a classifier with approximately 60 estimators, before steadily increasing the number up to 100 estimators until a model would pass the test. The lower and upper bound on the number of estimators was set based on an initial test comparing models build using between 10 to 140 estimators. The

models started to consistently pass the test at around 50 estimators, while rarely failing at around 100 estimators. A lower and upper bound of 60 and 100 estimators respectively was therefore considered reasonable. The ability to save and reuse the same model in the Scikit-Learn library was not used. Instead, a new classifier was build for each screen capture session. The reason was always to have the ability to increase the training and test set without having to save the model. Indeed, this is not optimal in terms of consistency and a classifier should have been built and comprehensively tested and reused. Other approaches for choosing this and other hyper-parameters as well could have been based on using scikit-learns cross validation tool.

Prediction of time requires 9 correct classifications of digits in order correctly classify the time, and it is required to work for all the 24 hours. However, to speed up the reading of timestamps the opportunity to set the time resolution was build into the timestamp reading module. This way, knowledge about the current time could be used to shorten the prediction time by limiting the number of digit classifications. For instance, if the simulation lasted only 2 hours, it would be required to classify timestamp with hourly resolution only for half of the time, effectively limiting the number of required classifications of digits to 5 digits for half the simulation.

Other approaches for classification of digits were also attempted:

- 1-Nearest Neighbour based on summing binary values in 84x84 binary pixel image. Digits representing 8 and 9 was to close in terms of the number of pixels to give reliable classification in the presence of cropping.
- Tesseract OCR for text recognition. Successfully read the timestamps, but was to slow as it used.
- k-Nearest Neighbour on 12x12 binary pixel image did not achieve sufficient accuracy.
- Support Vector Machine on 12x12 binary pixel image did not achieve sufficient accuracy.
- Skikit Learn's handwritten digit dataset was used to increase the volume of the dataset and to make the models more robust, but it decreased the performance of the Random Forrest Classifier models.

It could also have been attempted to build a classifier based on pre-extracted features from the binary images of the digits. For instance, high-resolution images of a digit could have been divided into multiple different predefined grids with different cell shapes, before summing the binaries of the resulting cells. This feature extraction approach could have been implemented either by using predefined cells or approximated using multiple different kernels for re-sizing the image. However, the Random Forrest Classifier based on the 12x12 pixel image was more than sufficient enough in terms of speed and accuracy. On average, the classifier obtained a prediction time of 4.4 milliseconds per digit.

4.2.3 Mask extraction

Based on pairs of sufficiently synchronized target and source images, the source image was used to extract masks of the objects in the source image. As briefly mentioned in earlier in the intro of Section 4.2, this task was about clustering pixels belonging to the non-background dominant colors in an image, and can be divided into mainly three steps in addition to noise removal and smoothing of edges. For an overview of the mask extraction method, we refer to the intro and Figure 4.3. This section walks through each of the steps and ends with a short discussion about possible extensions and improvements.

Thresholding and Closely Connected Spatial Region Dissection

The first step is about removing as much of the background as possible through custom thresholding in the HSV color space, depending on the distribution of pixel values of uniformly colored vessels.

It was discovered that the simulator did not use strong colors for the background such as the sea or coastal areas, while the color of uniformly colored objects would get high values in the value and saturation dimension of the HSV color-space. This can be seen in Figure 4.6. This simulator behaviour made the pixel values separable, such that it was possible to remove the background by a suitable thresholding scene. In addition, the blue spectrum of the hue dimension was removed by thresholding since the background naturally would contain a lot of blue colors due to characteristics of the maritime environment. This fact is the reason that the blue range of the color spectrum was avoided when replacing the texture with uniform colors. However, the task became harder with high intensity of en-

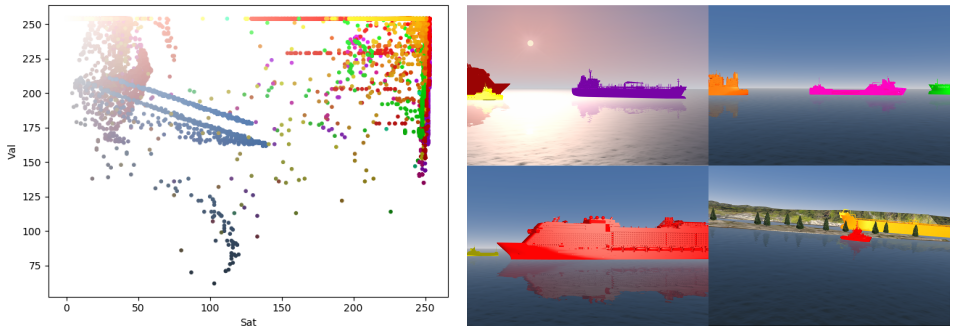


Figure 4.6: K-Sim 3D-view of source images with no environmental noise and their common HSV pixel distribution. The scatter plot shows that vessel colors are separable from the colors of the sea, sky, and sun, while the environmental colors and vessel colors are weakly mixed.

vironmental noise, which would mitigate all colors, and in the presence of coastal areas with building in the background that would contain many different colors. In Figure 4.7 it is seen that buildings in one of the images have orange rooftops and that the corresponding pixels likely blends with the vessel colors. This background interference was a source

to mask error, but the section later explained how this was handled by a change in the bandwidth hyper-parameter in the hierarchical clustering. Nevertheless, it illustrates the importance of setting the thresholding limits properly and why it was advantageous that a sequence of source images has more or less the same distribution of colors. This background interference was also the reason for the importance of planing the environmental noise throughout a simulation beforehand. From the figures, it can be seen that a combination of the distributions in the two scatter-plots would have made the background pixels and the object pixel inseparable without compromises on the cost of threshold accuracy.

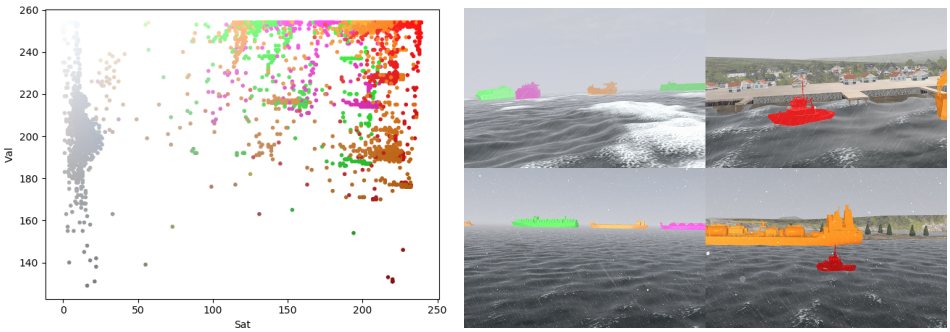


Figure 4.7: K-Sim 3D-view source images with a high degree of environmental noise and their common HSV color-space pixel scatter-plot. The scatter plot shows that the colors of the sea are separable from all other colors, while the environmental colors and the vessel colors are mixed.

A script was written to visualize the distribution of pixel values in order to determine suitable thresholding values and to do the subsequent thresholding. Thresholding was done by combining multiple rectangles with one corner attached to the maximum values for saturation and value. The height and the width of these rectangles were then chosen based on the visualization such that dominant colors from vessels were kept while the background colors was removed. In addition, the blue spectrum was also removed. The removal of the blue spectrum was also beneficial for the mask extraction step, as will be explained in the following section. An illustration of a thresholding scheme is shown in Figure 4.8.

The thresholded images were then dissected by splitting groups of objects that were not closely connected. By closely connected regions, it is meant that two or more regions are spatially close to each other without actually being connected. This region dissection was done by first expanding groups of pixels through morphological dilation, and then computing the connected components, before using the arithmetic bitwise-and operation between the connected component masks and the original thresholded image. This way, the image was dissected into multiple segments, each containing a closely connected group of pixels. This process is further denoted as closely connected spatial region dissection or only region dissection. There were two reasons for doing so. The first reason was that overlapping vessels and elements in coastal environments could divide the vessels into multiple parts, and it was desired to connect these divided parts of vessels as to not get multiple close masks of the same vessel. Size of the pixel segments was not taken into

consideration, which is unfortunate since groups of vessels far away naturally require a different magnitude of dilation compared to vessels at close range. This aspect could have been implemented simply by using a different number of iteration in the dilation transform based on the segment sizes but was not considered before after having completed the use of the data acquisition method. Therefore, it proposed as a possible future improvement. The second reason was to ease the subsequent clustering process by separating regions having objects of similar colors, and becomes apparent in the following section.

The thresholding scheme is characterized as being very simple and not at least dependent on an error prone manual effort for tuning the thresholds. There exist multiple ways of solving this task since it is about separation in its nature. For instance, a set of labeled images with representative distribution could have been used for building a SVM. However, a natural solution would have been to embed the thresholding task into the following VBGMM clustering. Given a set of clusters obtained from VBGMM, background colors could have been threshold based on the saturation and value centres of the clusters. This embedding would have required an extension of the existing methods, and is disscussed in the end of the following section.

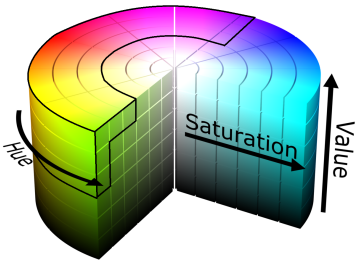


Figure 4.8: Illustration of a thresholding scheme. Strong colors are kept, while weak colors and the blue spectrum is removed by thresholding

Bayesian Gaussian Mixture Model Clustering

An image with the resolution of 1394x924x3 would have 1 288 056 data points per channel, and after having thresholded the source image and dissected the resulting image into closely connected regions, these regions could contain a massive number of data points. The clustering method had to be able to handle such a large amount of data in each region within a reasonable time, as well as finding the unknown number of true color regions. While hierarchical clustering or other unsupervised clustering methods would have been able to identify the correct number of regions, they would require to much time to be useful. Furthermore, the clustering method had to be able to cluster objects that could consist of closely non-connected parts, and that was highly concave, while still considering the spatial location of the object. Separation of different regions not being closely connected relaxed the spatial dependency requirement because it narrowed the spatial region. Still, clustering was based on both color similarity and pixel proximity.

The solution was a two-step clustering method followed by region dissection of resulting clusters. First, probabilistic clustering based on VBGMM would find sub-clusters based on color similarity and spatial proximity, and by such reduce this information adequately considering that GMM have the capability of representing distributions in vast amounts of data and is computational effective due to its Gaussian nature. In addition, since variational learning only needs an upper limit for the number of components and finds the optimal number by suppression, those components that don't contribute, it reduced the

effort in the next step. Next, agglomerative hierarchical clustering would combine these sub-clusters based on the hue color distance between the centres of these sub-clusters. Finally, since distinct objects of similar colors may have been connected by a third but now separated object, region dissection would split clusters if they had two segments of similar colors being sufficiently spatially separated. Seen in retrospect, the combination of probabilistic clusters by hierarchical clustering made the spatial variable as input to VBGMM redundant, led only to increased runtime. However, if the range of object had been taken into consideration in the context of region dissection, the spatial information would have been necessary for being able to adjust the magnitude of dilation of regions, based on the size of connected regions being made up of probabilistic clusters.

The Scikit-learn[8] software library provided an algorithm for VBGMM. The hue color in the HSV color-space and spatial location of each pixel was used as input variables. The maximum number of components was set to 25 and the maximum number of iteration was set to 350. No other parameters were set differently than the default values set by the implementation. A subset of 8000 datapoints was chosen for fitting to limit runtime, a small number compared to the maximum number of possible datapoints. However, since the centre of the red spectrum is equal to 0 degrees, it is numerically split in half since the color spectrum wraps around from 360 degrees to 0 degrees. This orientation of the hue-dimension would affect problems for the clustering if not appropriately handled. The solution was to exploit the fact that the blue spectrum with centre at 240 degrees had been removed thresholding, by rotating the hue-dimension by 100 degrees and placing the blue spectrum at 0 degrees. Each closely connected region in thresholded images was then clustered by fitting a VBGMM to the input datapoints and then classifying the rest of the pixels. The clustering is visualized in Figure 4.9, and we can see that each region and each vessel contains multiple clusters. These sub-object segments are the motivation for applying another layer of clustering for combining these different clusters such that they constitute semantically meaning-full objects. Another illustration is shown in Figure 4.10, and supports the need for a step or sub-process that extracts the different spatial color components from objects.

The main drawback from only using the hue dimension in the HSV color-space is that it is not robust against insufficient thresholding. If pixels were to pass through, the clustering would not have been able to distinguish between them and pixels belonging to objects. Even worse, if the background pixels and object pixels were to be inseparable by thresholding but still belonging to different distinguishable distributions, the capability of the GMM for representing these distributions would not have been utilized. By using the saturation and the value dimensions and a suitable method for combination of clusters, it should be possible extend the method as a whole to separate and remove background pixel clusters from and object pixel clusters, and by such relaxing the requirements for accurate thresholding or even removing its current need. This alternative way of thresholding should therefore have been implemented between the probabilistic clustering step and the subsequent hierarchical clustering step. In addition, since background colors tend to dominate the image, it could be wise to address the issue of the unbalance between object and background colors, such that the majority of Gaussian components are spent on object colors; Bias selection of datapoints, smoothing of weak colors, and proper initialization of algorithm are examples of suitable efforts. All these and other possible extensions or

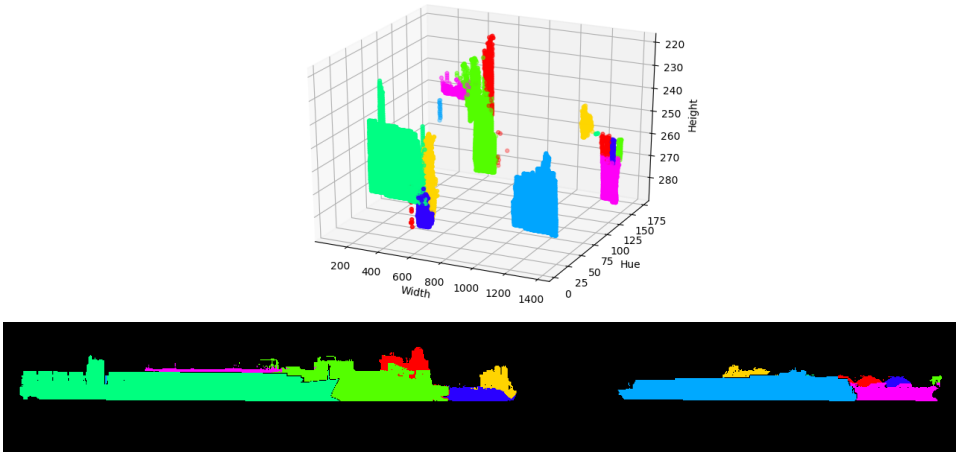


Figure 4.9: Visualization of clustering based on hue similarity and spatial proximity in a thresholded K-Sim source image using VBGMM

improvements are not discussed any further but is instead proposed for future development on the topic of mask extraction from images with distinguishable distributions of strong and dominant colors. The reader is encouraged to keep it in mind in the following sections. The proposal is also given in Chapter 8. We continue by covering how the clusters are combined by hierarchical clustering in the following section.

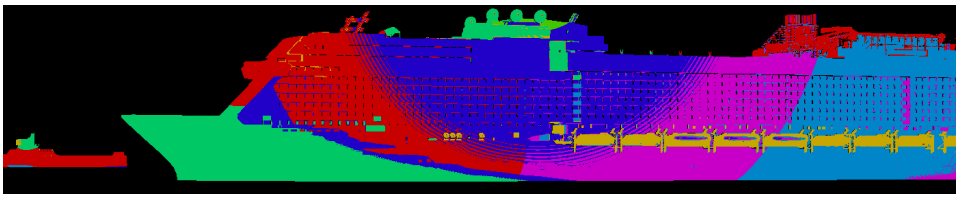


Figure 4.10: Visualization of extracted components of objects by clustering based on hue similarity and spatial proximity using VBGMM. The circular pattern is due to light-conditions caused by the sun in the simulations, and gives strong color variations despite the object having an uniform color.

Hierarchical clustering

At this point, the pixels in closely connected regions of thresholded pictures would have been clustered based on their probability of belonging to different Gaussian distributions in a GMM based on hue and pixel location. However, multiple distributions would, as shown in the last section, constitute different segments of one object. Therefore, different clusters had to be combined to assign each pixel to a semantically meaningful object.

This combination was done by agglomerative hierarchical clustering (see Section 2.3.2)

using the Manhatten metric for distance measure and weighted generation of new centres, based on hue mean centres of distributions. The threshold or maximum distance between the centre of two clusters before they were considered two final distinct clusters were set empirically from testing. A value equal to 10 was found to give good results for images without any environmental noise, which was the light conditions that would spread the uniform color of the objects the most. For images with some intensity of environmental noise, a lower value was sufficient. Besides, a lower threshold value allowed for increased separability between objects of similar colors and between objects and coastal background.

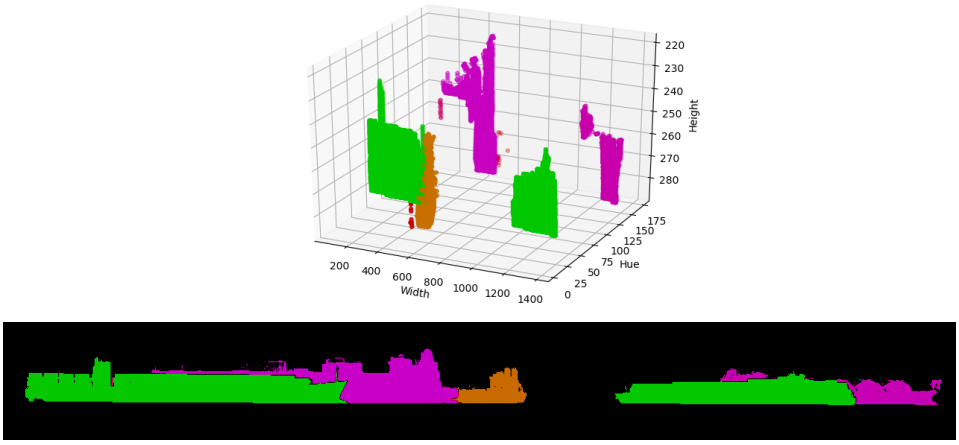


Figure 4.11: Visualization of probabilistic hierarchical clustering based on hue similarity and spatial proximity in a thresholded K-Sim source image using variational learning of Gaussian mixture models and agglomerative hierarchical clustering.

It was also considered to use mean spatial centres and their relative proximity for connecting the different distributions based on a proximity closeness measure. This option could have made the clustering more robust and also allowed for an increased color similarity between distinct objects in close proximity. By using the covariances between spatial centres, different sized vessels could have been into account as well. Nevertheless, hierarchical clustering along hue dimension proved to be sufficient, and therefore this was not attempted.

Closely connected spatiel region dissection of clusters

After having clustered the different hue mean centres of the distributions, the closely connected regions were split into multiple clusters. However, if a third object had connected two different objects with the same color, these would now have been combined into a single cluster. To deal with this possibility, the resulting clusters were attempted connected using morphological dilation and connected component calculation in a similar fashion as the last time. If they where close enough they would end up being connected and thus combined, and if not they were split into two clusters. Again as in Section 4.2.1, the spa-

tial size of the objects was not taken into consideration, and therefore objects at different ranges was not discriminated differently. This option could as mentioned have been implemented by using a different number of iteration in the dilation transform based on the size of the pixel segments, but it was also not thought of before after having completed the data acquisition.

Morphological noise removal and edge smoothing

The final extracted masks would contain noise is the form of grains and holes due to either insufficient uniform coloring, inaccurate thresholding, or environmental noise. Besides, the masks would have rough edges, which was considered undesired. Morphological closing was therefore used to smoothing the images, as seen in Figure 4.12.

(a) Before smoothing by morphological closing transformation

(b) After smoothing by morphological closing transformation

Figure 4.12: Visualization of smoothing of masks by morphological closing transformation, and final masks after applying the mask extraction method.

Time consumption

Measurements showed that it took 25 minutes and 35 seconds to extract masks for 285 images that, on average, contained two to three objects at a close to a moderate distance. The time per image was highly dependent on the number of vessels and the corresponding number of pixels in the image. An image with a single and small vessel could require only 0.5 seconds to process, while an image with multiple vessels or a single huge vessel could require up to between 12 and 15 seconds to process.

4.2.4 Mask assessment and adjustment

The steps so far in the mask extraction method would have produced masks for each pair of target and source images. However, there are multiple sources of mask errors, such as insufficient thresholding leading to false positive masks, or displacement or rotation between objects in the target image and the extracted masks due to synchronization error. Therefore, there was a need for manual assessment and adjustment of the extracted masks. A script was written to enable this need with the following features:

- Rejection of specific masks, to remove false positives.
- Horizontal and vertical adjustment of all masks simultaneously, to overcome displacement. Masks connected to the boundary of the images was padded using the edge of the mask if adjusted into the image by.
- Repeating extraction of masks with a different bandwidth within ± 3 around 12 in hue, in case of having multiple masks per objects due to a large variation in hue, or having a single mask covering multiple objects or elements.

The step is illustrated in Figure 4.13, where multiple false positive masks are rejected and the remaining three masks are adjusted.

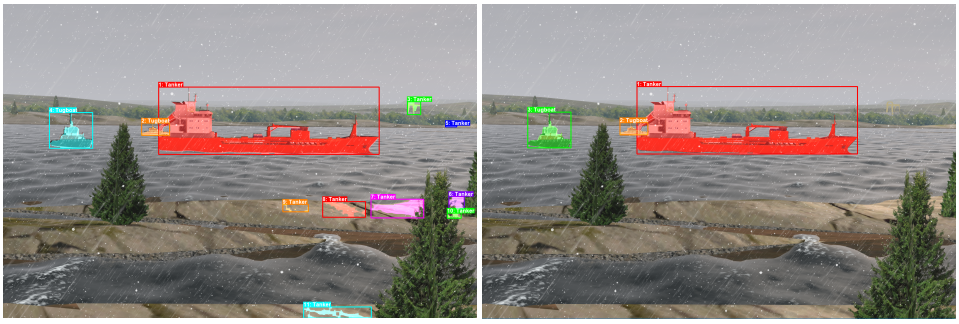


Figure 4.13: K-Sim target image with masks before (left) and after (right) assessment and adjustment. The image before assessment of masks has both false positive masks and displaced masks. False positive masks are removed, and remaining masks adjusted simultaneously both horizontally and vertically.

The script would take a collection of instance-labelled images with potentially false positive or inaccurate masks, and make it possible to process them one at the time. During the creating of the synthetic dataset, measurement showed that it took 1 hour and 55 minutes to assess and adjust 922 images in which 819 images were kept. These images contained none to moderate levels of environmental noise.

4.2.5 Time efficiency and memory load minimization

Since the mask extraction step was computational demanding but could run automatically, while the others required manual labour or supervision, the pipeline was divided into four parts that could partly be implemented simultaneously. Firstly, simulations could be carried more or less without supervision once the setup was ready, which was beneficial as the simulator runs in real time. The same applied for the process of sampling synchronized pairs of images, except that the speed is twice as high. Secondly, given that there existed a dataset with assessed synchronized pairs of images, its masks could be extracted automatically. Thirdly and lastly, either the manual assessment of synchronized pairs of images or the manual assessment and adjustment of extracted masks could be done simultaneously with the two other tasks. This setup was highly effective in terms of time consumption

because it allowed for concurrent work when creating data.

Furthermore, masks were saved as binary images with a size equal to the bounding box implicitly given by the mask. The filename was used to encode the location of the mask relative to the width and height of the corresponding target image. It was necessary to save the masks as binary images for preserving the accuracy of the masks, as other approaches such as saving masks as polygons would have been insufficient. Moreover, a size equal to the bounding box saved memory, essential during the training of machine learns to save RAM, besides easing file transferring and storage.

Chapter 5

Experiment

Based on the method for acquiring synthetic labelled data from K-Sim that was developed and explained in the last chapter, we will in this and the next chapter cover an experiment that was conducted to continue exploring the practical implication of using synthetic data for instance segmentation in the maritime domain, as well as applying LIME to explain instance segmentation models. The first section of this chapter states the motivation and priorities for the experiments and give a brief overview of the different experimental steps. Then, the second section sequentially covers the different steps of the experiment. The results from the experiment is presented and discussed in the following chapters.

5.1 Experimental background

5.1.1 Motivation

The ability to use synthetic images and data from K-Sim could, as mentioned in the introduction of this thesis, be highly beneficial for development of computer vision models and autonomous systems in the maritime domain. This experiment has therefore been aimed to further study the implications of its usage. Since the development and implementation of a method for extracting instance-level labeled data from K-Sim took a considerable amount of work, some priorities have been made to focus on what was deemed the most interesting aspects of the use of synthetic data. Firstly, it was of high interest to see how well models trained purely on synthetic images would perform in the real world, and how the use of synthetic images affected performance of real-world adapted models. Secondly, it was of interest to see if the models would have the ability to distinguish between similar vessels, and how synthetic images would affect this ability. Thirdly, it was of interest to see how different domain adaptation techniques based on augmentation and different fine-tuning approaches would affect performance. Lastly, it was of interest to see if LIME could offer

some explainability for the performance of trained models.

Furthermore, the experiment has been based on the Mask R-CNN instance segmentation model as stated in the problem description (see Section 1.2). It is among the state-of-the-art instance segmentation architectures [17], and it still serves as a basis for further development of architectures within instance segmentation. Besides, a popular open-source implementation by Matterport [12], which uses ResNet101 in the ResNet-FPN backbone for feature extraction, makes the architecture easily available. Therefore, the architecture a good basis for research within the topic of instance segmentation.

5.1.2 Overview

An instance segmentation task was defined as: *Detect, classify and predict masks of four different vessel classes: tankers, container ships, LNG carriers, and tugboats.* Mask R-CNN was to be used for training models to achieve this task.

The preparation before the training av models consisted of five separate tasks:

- A popular implementation of Mask R-CNN, implemented using Keras and Tensorflow, and based on a Feature Pyramid Network (FPN) and a Resnet101 backbone, was obtained from Matterport's open-source repository [12]. The implementation was then configured for the instance segmentation task as it is defined above.
- A synthetic training dataset and a synthetic testing dataset from distinct simulations in K-Sim was acquired using the method developed and explained in Chapter 4. In addition, a short video was created to be used for additional visual assessment.
- A real-world training dataset was created by collecting images from two vessel image repositories[46, 47], and a real-world testing dataset was created by sampling 46 frames from a video that was put together from multiple video clips gathered from tree youtube channels [48, 49, 50] plus an additional 26 images the same image repositories. The training and testing dataset was then manually labeled using VGG Image Annotator (VIA) [2, 3]. Due to reasons that will be given in Section 5.2.9, the video-frames in the testset were used as the validation set.
- The training dataset was doubled in size by augmentation based on random zoom, random rotation, and cropping. It was then split into a training and validation set with a ratio of 2220:80, before images in the training set which was related to images in the validation set though augmentation was removed. Final split the became 2061:80. Justification for pre-training augmentation will be given in Section 5.2.4. An augmentation scheme to be used during training for regularization and for decreasing the dataset bias between the synthetic and real-world dataset was then established.
- Two different domain adaptation methods for adapting a synthetic model to the real-world domain based on the use of the labeled real-world dataset were established. They consisted of two different training methods and a common data augmentation scheme to increase the diversity of the real-world dataset for the two methods.

An overview of the datasets are given in Table 5.1. Augmentation was implemented using Imgaug [51], a popular library for image augmentation, in addition to custom scripts for the pre-training augmentation.

Six different Mask R-CNN models where then trained with the task that was defined in the beginning of this section. Two pure synthetic models, two adapted models from the synthetic domain to the real-world domain, and two pure real-world models. Fully fine-tuning and gradually fine-tuning was decided to use as fine-tuning approaches, to see how no freezing and gradually unfreezing layers would affect performance. A list of the six different models is given below:

- A fully fine-tuned synthetic model
- A gradually fine-tuned synthetic model
- A fully fine-tuned adapted model
- A gradually fine-tuned adapted model
- A fully fine-tuned real-world model
- A gradually fine-tuned real-world model

Fully fine-tuning was done in one stage, corresponding to the last stage in the gradually fine-tuning approach. Gradually fine-tuning was done in three stages:

1. The heads of the Mask R-CNN architecture was opened for training, that is classification, bounding box, mask header and RPN.
2. All layers following the fourth layer in Resnet101 was opened for training.
3. All layers were opened for fine-tuning, with 1/10 of the original learning rate used at the previous stages.

The two pure synthetic models and the two pure real-world control models were based on a Mask R-CNN model pre-trained on the MS COCO dataset, resulting in a transfer learning problem due to a change in domain and label space. The best performing model among the two synthetic models on the synthetic testset was chosen as a target model for the real-world domain adaptation. In addition, the two pure synthetic models and the two pure real-world models would work as control models to see if domain adaptation took place, and to see what effect the use of synthetic data had. The same augmentation scheme was used for training both for the the adapted models and the real-world models.

All training was done using the free-of-charge cloud-based service Google Colaboratory which is based on Jupyter Notebooks. Training of each of the models lasted between 12 hours and 2 days, as the length of the training determined by early stopping based either on stagnation in validation loss or overfitting characterized by increasing validation loss together with decreasing training loss. The performance was measured using mAP (see Section 2.4.2), and was visualized and monitored using the visualization tool Tensorboard

which comes with Tensorflow. Epoch size was varied dependent on the need for frequent calculation of validation scores.

Additionally, it was attempted to get explanations for the prediction from Mask R-CNN by the use of LIME (see Section 3.2). Two types of explanations were attempted obtained; one explanation for the class probability score, and one explanation for the mask prediction only based on mask pixel-wise IoU (see Section 2.4.2).

	Synthetic			Real-World		
	Train	Val	Test	Train	Val	Test
Images	2061 (1031)	80	149	151	45	71 [26]
Tankers	2071 (1047)	84	68	82	16	24 [8]
Cargo ships	1946 (984)	81	75	77	15	24 [9]
Tugboats	632 (326)	32	58	69	18	25 [7]
LNG carriers	1522 (769)	57	64	78	14	25 [7]

Table 5.1: Synthetic and Real-World datasets overview. The numbers in parentheses are the numbers in the original synthetic dataset before pre-training augmentation. The numbers in square brackets are the numbers in the additional real-world testset which was not used for validation.

5.2 Method

5.2.1 Synthetic dataset generation

Based on the method for acquiring synthetic labelled data from K-Sim that was developed and explained in the last chapter, a synthetic training and a synthetic testing dataset was created from distinct simulations.

The number of different classes was limited to four, while the original plan was to use all 25 available classes of vessels. To simultaneously test the robustness of the mask extraction step of the data acquisition method, which relied upon the color encoding of classes, it was planned to increase the number of classes progressively. However, due to time consumption during the creation of the dataset, it was determined to limit the number of classes to only the four that were chosen from the start. However, this was considered a possible outcome, and therefore, the initial first classes were chosen such that the final dataset would both have similar and distinct classes of vessels in case the number of classes became restricted to the initial classes. The final classes were tankers, container ships, LNG carriers, and tugboats.

The entire process of creating the dataset lasted approximately ten days, and a total of eleven simulations were created. Nine simulations for creating the training dataset, and two simulations for creating the testing dataset, respectively. In addition, a video was created to be used for an additional visual assessment of the performance. The majority of the time during these ten days were spent on creating and carrying out these simulations and taking synchronized screen captures from the recording. A total of approximately 8

hours were used on the mask extraction step and mask assessment and adjustment step, in addition to the tuning of the hyper-parameters. Besides, much time was spent on regular file handling. The final dataset contained 1261 instance-level labelled images with a total of 3645 vessels.

Each of the simulations used to create the training and testing datasets had different objectives in terms of adding diversity. For instance, the first simulations contained only aligned non-overlapping vessels with their side towards the viewpoint route, while the fourth simulation contained clusters of overlapping vessels with a variety of different heading. Simulation seven, eight, and nine all contained a moderate to a high level of environmental noise, with simulation eight having a mix of overlapping and non-overlapping vessels. Simulation four had a city and harbour in the background. In the two last simulations used for creating the testing dataset, it was attempted to capture all the diversities that was thought to be in the training dataset. While the simulations contributed towards diversity, it is clear that there exist numerous additional ways to add diversity, such as having overlapping vessels from the same class. During the mask assessment step, only masks of vessels that were considered possible to classify was kept. That is, masks for barely visible vessels due to either being partly hidden by other vessels or coastal environments or cropped out of the image was rejected. Some images were also rejected due to containing vessel parts that were considered impossible to classify for both human and machine learners. The training dataset ended up heavily unbalanced in terms of the number of different vessels between the classes, as seen in Table 5.1. Unfortunately, this was not thought of before after having started the training of the models. We delay the discussion of the implications of the imbalance until the discussion in Chapter 7. A sample of synthetic images with their masks from the synthetic dataset is shown in Figure 5.1.

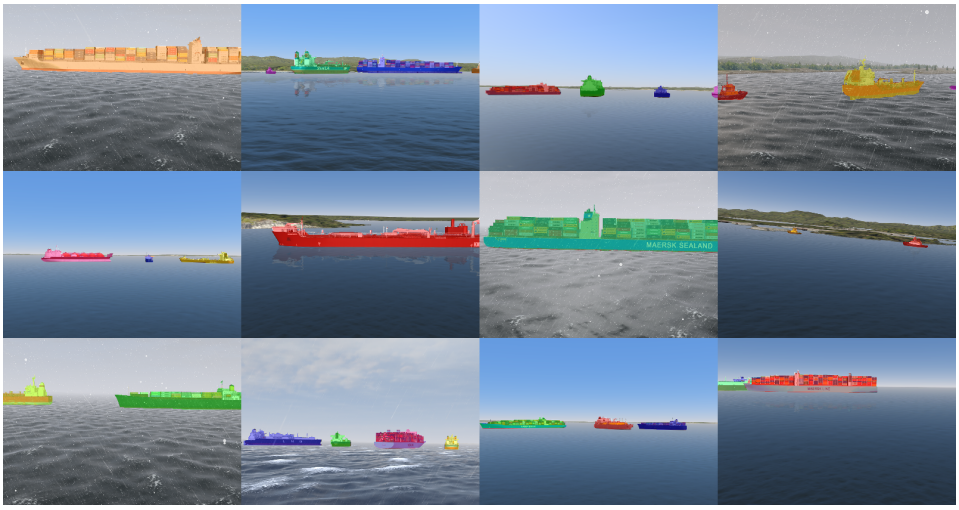


Figure 5.1: Samples of instance-level labeled synthetic images

5.2.2 Real-World dataset generation

Two different approaches were followed in order to create a training set and testing set. First, a collection of images were gathered from mainly ShipSpotting.com and some from Vesseltracker [46, 47]. The images were if necessary re-sized to less than or equal to 896 in width. Then, a collection of clips from videos were gathered from Vesseltracker and three YouTube channels [48, 49, 50], and put together for a testing video, before a total of 50 frames were sampled periodically and re-sized to a width of 896 pixels. The images were used in the training dataset, and the frames were used in the testing dataset. They were then labelled using VGG Image Annotator [3]. This process resulted in 151 training images and 45 testing images with a total of 306 and 63 vessels, respectively. However, since the testing images was determined to be used as validation images as well, due to reasons that is explained in Section 5.2.9, and the fact that 45 testing images was considered to provide an insufficient statistical significance, it was determined to create an additional 26 labeled testing images. Therefore, the total size of the testset became 71 images with a total of 98 vessels. All the images required approximately 9 hours and 50 minutes of manual labelling effort. Compared to the 3645 vessels contained in the synthetic dataset, which required approximately 8 hours to label, the difference is captivating. All the images and the video were collected over approximately one month.

Caution was taken for not having common ships in the training and testing dataset, as many images found online would be screen captures from videos. Images with huge and detailed-rich harbours or other dominating coastal environments in the images were attempted limited, while breakwaters, shores, or coastal environment at far ranges were included. This caution ensured a similarity between the synthetic and the real-world images. It was also attempted to keep light condition the same between all the images, and between the images and the video.

A sample of instance-level labelled real-world images is shown in Figure 5.2, and shows all the four classes of vessels.

5.2.3 Dataset biases

The synthetic training dataset and the real-world training dataset consists of 1030 and 151 images, respectively, with four different classes. Both these numbers can be considered a small number of images in the context of deep learning. Therefore, there was a risk of overfitting, or loss in generalization ability, should the training last too long. Since the experimental plan did not allow for comprehensive training and testing due to time constraints, it was important to identify sources to biases that could contribute towards overfitting, and which could be encountered by appropriate data augmentation techniques. Here, we shall specify what was thought to be the different types of sources to different biases between the synthetic and the real-world maritime domains. These sources are referred to in the following section about data augmentation and training. The different sources were thought to be:

- Computer graphic bias:

Figure 5.2: Samples of instance-level labeled real-world images

While today's real-time 3D-engines can produce highly photo-realistic graphics, it is often limited due to hardware restrictions. Light conditions can be limited in terms of reflection and shadows to reduce the computational cost, and texture and scenery can be kept simple to reduce memory load, among other aspects. Real images will contrastively, will be affected by, for example, camera devices, lighting conditions, blur due to shaking or bad focus, and lens occlusion. These aspects introduces numerous sources to a covariate shift (see Section 2.5). Moreover, the different rendering techniques could introduce features that would not be present in real-world images, which in turn can be caught by deep learning.

- Sampling selection bias:
  The content and diversity of the synthetic dataset are highly dependent on the setup of the simulations, which affected the number of object instances in each image, the range of objects, the environmental noise, the coastal environment, the orientation of objects, and the viewpoint perspective, among other aspects. In addition, K-Sim allows for all possible viewpoint. Contrary, real-world photographers and perspectives are often limited to harbours, breakwaters, other vessels, and occasionally aircraft. The ability to orchestrate the collection or orientation of maritime objects is also restricted or non-existent. The real-world dataset used in this experiment is based on photos and video footage from mainly ship spotters at harbours or shores. This difference introduces a sampling selection bias (see Section 2.5) between the synthetic dataset and real-world dataset due to a difference in the distribution of domain-invariant features. In particular, there is a significant difference in the distribution of range from camera and objects between the datasets. The same goes for the number of objects in each image and the overlap between them. These features can be denoted as geographical features, and are a significant source of bias.
- Compression bias:

Images and videos collected online are affected by digital image procession such as editing and compression techniques for storing. This would introduce features that would not necessarily be present in synthetic images.

- Instance-level mask bias:
  Since the labelling method of the synthetic and real-world datasets was different, a difference between the two types of masked was observed. While the masks of objects in the synthetic data were characterized by having a high amount of details, the masks of objects in the real-world were characterized as blobs around objects. The smoothing of the masks in the synthetic data had reduced this difference on some level, but it was still clearly present.

5.2.4 Synthetic dataset augmentation

In order to close the gap between the synthetic and real-world datasets and to add regularization effort, the synthetic training dataset was augmented both before training and during training.

Due to having observed a scene bias in terms of the size, range, and rotation of objects and sea surface between the synthetic and real-world dataset, it was desired to apply random zoom-in and random rotation. However, due to the size of the objects, there was a high risk of cropping objects due to zoom-in to a degree where they would be barely visible. Moreover, the random rotation would naturally and occasionally lead to triangles of empty pixels at the corner of the images, and to amplify zoom-in, it was desired to crop away as much of the image containing empty pixels as possible. This cropping would also reduce the chance of having the neural network learning the rotational transformation between original and rotated images, which could have removed the effect of the rotation. Nevertheless, the former was the primary motivation.

The reason why the training set was augmented before training, was due to the fact it was not found a way to fully augment the data as desired during training because of restriction in the augmentation abilities. In addition, when attempting to do minor random rotation and cropping, there was complains from the Mask R-CNN implementation about changes in mask shapes, which eventually became fatal such that the training frequently crashed. It is thought that this would be been possible to fix, but it was not considered at the time when the training setup was implemented.

Therefore a script was written to randomly zoom in on objects, randomly rotate images, cropping away empty pixels, while simultaneously avoiding exaggerated cropped masks by either entirely cropping out objects or restrict cropping to the boundary of the masks. In addition, the script would re-size all images to having a maximum width of 896 pixels. The reason was that it was determined to limit the maximum dimension to 896 pixels, based on the fact that the real-world images mostly had a width between 600 and 800 pixels, except those few having a width up to 3000 pixels. The dimension of 896 is divisible by 32, which was required because of the FPN in the Mask R-CNN backbone. Re-sizing reduced padding of small images during training, and the re-scale effort on the synthetic images. Re-sizing before training was also desirable as to limit file transferring time during the

setup of training in the cloud-service Google Colaboratory and during training for limiting memory load on GPUs. The script was then used to first augment the whole dataset, before re-sizing it. Then the original dataset was re-sized. This transformation effectively doubled the size of the training dataset while saving memory due to file size. Re-sizing was done after cropping in order to avoid as much loss in image quality in order to limit the degradation of image quality due to re-sizing.

The training dataset was then split into a training and validation set with a ratio of 2141:80 at first, before the images in the training set that was related to images in the validation set by augmentation was removed. This removal avoided a possibility for obtaining an artificially low validation loss based on the presence of similar images between the two sets, but at the cost of less data to use for training. The final ratio then became 2061:80, which means that 79 images were removed from the training set. The reason why only 79 images and not 80 images were removed, is likely due to an image in the validation set not having been augmented beforehand due to the caution for not losing a significant presence of objects due to random zoom, rotation and cropping. Thus an image in the validation set likely did not have an image counterpart in the training set which could be removed.

Then the augmentation scheme to be used during training was created by the use of a comprehensive image augmentation library called Imgaug [51]. Below is a short description and explanation:

- 50 % of the images was flipped.
- 70 % of all images was augmented with 1 to 4 filters among 4 different filters: *Gamma and contrast* was used to simulate overexposure, equal *add to colors* to add darkness or brightness, per channel *add to colors* in RGB color space and *add to hue and saturation*, to gain an extended range of colors. 5 % of all images not being augmented by these one or more of these filters, was overlayed by the gray-scale variant by between 40 % and 100 % to extend the range of color intensities.
- 50 % if all images was augmented with 1 to 2 filters among 6 different filters with different purposes. *Emboss* and *sharpening* was used for simulation shadows and highlight shapes and edges. Simplex noise, Gaussian noise and Gaussian blur was used to make the synthetic images closer to the real-world.
- 50 % of all images was augmented with 1 of 6 occlusion filters such as black, white, pepper-salt, and snowflake spots and squares. They were added as a regularization effort.
- These four different types of augmentations schemes was used in random orders for each image.

The augmentation scheme would augment as much as 93.75 % of all images to some

degree when excluding flipping, based on:

$$\begin{aligned}\Pr(A \cup B \cup C \cup D) &= 1 - \Pr(\bar{A} \cap \bar{B} \cap \bar{C} \cap \bar{D}) \\ &= 1 - (\Pr(\bar{A}) + \Pr(\bar{B})) \cdot \Pr(\bar{C}) \cdot \Pr(\bar{D}) \\ &= 1 - (1 - 0.7 - 0.5) \cdot (1 - 0.5)^2 \\ &= 0.9375\end{aligned} \tag{5.1}$$

where A, B, C, and D are the probability of applying the different augmentations schemes respectively. A sample of augmented synthetic images are shown in Figure 5.3.

Figure 5.3: Samples of augmented synthetic images using during training of Mask R-CNN model.

5.2.5 Real-World dataset augmentation

The augmentation scheme of the synthetic data was based on the desire to close the gap in terms of bias between the two representation of each their respective domain, namely the virtual maritime domain of K-Sim and the maritime domain of the real world. On the other hand, the augmentation of real-world images attempted to increase the size and diversity of what could be considered a small dataset with few object instances per class. It was intended to be used for both the fine-tuning and the gradually fine-tuning adaptations techniques, as well as in the training of the two pure real-world control models. The augmentation intended to get as much out of the information about the real-world representation of the four classes and maritime domain, contained in the small dataset, without introducing features that are not present in the real-world or adding a too strong regularization effort. The latter could have contributed towards making the models under-fit to unseen target test data, by seizing too much of model capacity on non-existing features. Therefore, an augmentation scheme was restricted to flipping, geometric augmentation,

and mild change in colors, brightness, and contrast. The magnitude of the geometric augmentations allowed for a substantial decrease in the scale of the images, to obtain smaller objects. The reason was a *scene bias* between test and training images in terms of the range of objects. The majority of the images in the training set where close-up photos of vessels, while the test images created from video clips had vessels at various ranges, both short and long range.

An overview of the augmentation scheme is given below:

- 50 % of the images was flipped.
- 100 % of images was augmented with either:
    - Affine transformation with scaling, translation, rotation, and shear mapping, with either nearest neighbour or bi-linear interpolation.
    - Perspective transformation, padding, cropping, and rotation.
    - Padding, cropping, and rotation
- 100 % of images was augmented with 1 to 2 filters per image among 3 different filters: *gamma and contrast*, *add to color channels* both equally or per channel, and *add to hue and saturation* per channel.

This resulted in that all images where augmented to some degree. The setup for the augmentation scheme can be found in Appendix A in Listing A.2. A sample of augmented real-world images is shown in Figure 5.4.

Figure 5.4: Samples of augmented real-world images using during training of Mask R-CNN models.

5.2.6 Mask RCNN configuration and training

A configuration file for training of Mask R-CNN was inherited from an basic configuration file from the implementation, and customized from training.

The backbone was chosen as Resnet101. Another option would be to use resnet50, which would give a lower model capacity. Considering the size of the dataset, this could have been a good option. However, since the model would later be adapted to the real-world, Resnet101 with a larger model capacity was considered to be desirable. The implementation automatically takes care of embedding a FPN into the model architecture. The number of classes was set to 5; 1 for the background and 4 for the different vessels classes. Resize mode was set to "square" with max and min dimension set to 896 pixels, based on the justification given in Section 5.2.4. Images being of different size would be re-sized to having their max dimension equal to the max dimension given in the configuration file, and padded with empty (black) pixels to become squared. Max number of ground truth instances per image was set to 12. The use of mini masks to save memory load was allowed and set 112 pixels in both height and width. Learning momentum and weight decay were held equal to default values of 0.9 and 0.0001, respectively. Learning rate was set to 0.001 for the two first steps of the gradually fine-tuning, and set to 0.0001 for the last step of the gradually fine-tuning and the fully fine-tuning.

The implementation allowed for a custom image size in the epoch and the validation step, which can be different from the size of the training and validation dataset, respectively. This option is for enables optional frequency in the calculation of validation loss, and adjustment of time spent on calculating loss and accuracy of loss in the validation step. Frequent validation and high accuracy help for monitoring the training progress which is done after completion of each epoch, but increases time spent on validation. The number of training and validation instances in the epoch and the validation steps was at any moment depending on the purpose of the training.

As a last step, some layers were required to be skipped and retrained due since the number of classes was different from the pre-trained MS COCO model: mrcnn_class_logits, mrcnn_bbox_fc, mrcnn_bbox, mrcnn_mask.

During training a total of five different validation losses had to be monitored due to Mask R-CNN having a complicated pipeline:

- rpn_class_loss: The Region Proposal Networks classification loss describing a binary class label prediction (being an object or not)
- rpn_bbox_loss: The Region Proposal Networks bounding box loss describing the distance between the true and predicted bounding box parameters of any class
- mrcnn_class_loss: Loss describing how confident the model is to predict the correct class.
- mrcnn_bbox_loss: Loss describing the distance between the true and predicted bounding box parameters.
- mrcnn_mask_loss: Mask average binary cross-entropy loss for masks heads

Early stopping was based on the mrcnn_class_loss, mrcnn_bbox_loss, and mrcnn_mask_loss, and a sixth validation loss being the combination of all five validation losses.

5.2.7 Google Colaboratory training setup

The authors of the Mask R-CNN paper [17] estimated that it took between one to two days to train the architecture on the MS COCO dataset when using an 8-GPU machine. To minimize the time used to train models, it was decided to use a cloud-based service. Amazingly, Google offers a free-of-charge cloud-based Jupyter notebook service which offers a NIVIDA Tesla K80 GPUGPU with 12GB RAM and 350GB memory and a TPU hardware accelerators. It comes with common machine learning libraries such a PyTorch, TensorFlow, Keras, and OpenCV pre-installed. It allows for Linux commands which enable arbitrary instalment of other python libraries, the use of git, and the use of zip-files. Since it is a Google service, it also can mount Google Drive.

However, there is a 12-hour session limitation, meaning that all activity is limited to that available time. Afterwards, all memory erased. Since Mask R-CNN training session may require more than 12 hours, it is simply not possible to use the service without having to save and back up the progress. This restriction is highly impracticable, since pausing after a 12-hour session can lead to unfortunate timing, in addition to a possible demand for intricate file management.

However, it was discovered that if two separate Jupyter notebooks were connected to the service at the same time, they would both run on the same hardware. This exploitation allowed for both accessing files during training and for running multiple processes concurrently. Moreover, Mask R-CNN saves model weights for each epoch and moderates a Tensorflow tfevent log file. As a result, this was utilized by writing a script that periodically took a backup of the most recent model weights and tfevent log files by saving copies to Google Drive. This backup scheme allowed for un-monitored training for 12 hours at the time. By using Google Drive, file management was done effectively by managing copies in the Google cloud-service. There where only two minor drawbacks. First, that training had to be restarted from where it was aborted. Second, since each model weight is approximately 450MB, and since Google Drive only allows for 12GB free-of-charge storage, the period between backups had to be large enough as to not lose significant training progress due to memory overload. Together, these drawbacks resulted in a minor loss of epochs due to the service shutting down before having taken the last backup. Regardless, the fact that it was possible to train for free on a powerful GPU massively surpassed the drawbacks.

Therefore, two separate Jupyter notebooks were created. The first configured and ran the training, by mounting Google Drive, cloning the custom git Mask R-CNN repository with code and training data, installing Mask R-CNN, moving any backup weight to the correct *logdir* location if the training was to be continued, before starting a training session. The second configured and ran the backup process. In addition, two scripts were implemented in order to monitor the GPU load and to monitor the training progression. The script to monitor GPU load can be found in Appendix A in Listing A.3. In order to monitor training,

a dashboard provided by Tensorflow's Tensorboard was used. However, since the training was running on a local network located at a Google server, the Tensorboard process had to be tunnelled to the local client computer. Tunnelling was enabled by a free service called ngrok [52], which would output a public URL, which exposed the local server. The script for implementing the use of Tensorboard through ngrok can be found in Appendix A in Listing A.4. The solution and explanation was based on a tutorial by Chengwei Zhang[53]

5.2.8 Training of synthetic models

Configuration and training was done according to the description in the experiment overview of Section 5.1.2, and training setup and configuration of sections 5.2.6 and 5.2.7, but adapted to the synthetic training set and the synthetic intra-training augmentation scheme.

The synthetic data was earlier split into a training:validation ratio of 2061:80, but since the Mask R-CNN implementation of Matterport allowed for an optional epoch size, it was varied depending on the necessity for frequent validation loss. In the early phases of the training that was characterized by progressive improvements in performance, an epoch size of 1031 was sufficient. When the progression was close to stagnation, the epoch size was lowered, making it easier to control the training progression by applying early stopping. Each step of the training lasted until they reached stagnation in the improvement of the validation loss. If a significant undesired increase in validation loss became present after stagnation, the training was re-started at an earlier point at stagnation to avoid effects of overfitting.

If considering an epoch size equal to 2061, that is the size of the training set, the fully fine-tuned model and the gradually fine-tuned model was trained for 35 and 41 epochs respectively. This means that the models by the end of the training had seen 71 085 and 84 501 images respectively, and approximately 93.75 % of the images would have been unique in some way due to data augmentation as shown in Section 5.2.4. Unfortunately, the data augmentation was without random zoom or random rotation due to the complaints during training about changes in mask shapes as mentioned in Section 5.2.4. Training progression over time can be seen in figures 5.5 and 5.6.

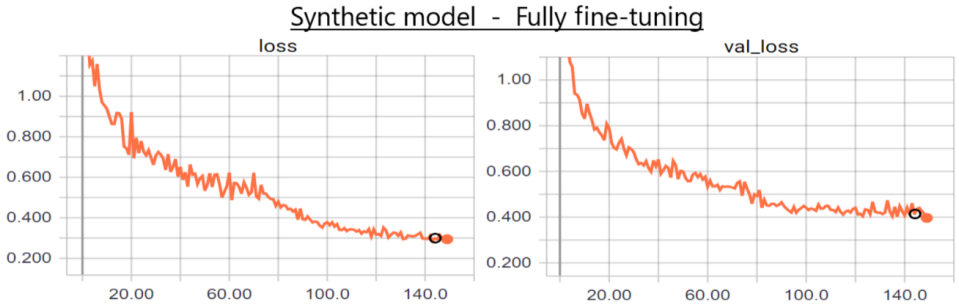


Figure 5.5: Training of fully fine-tuned synthetic model over time. Left shows training loss, right shows validation loss. The bend at epoch 76 is due to an increase in epoch size. Epochs (epoch size): 1 - 75 (220), 76 - 117 (1031), 118 - 150 (515). Model after epoch 142 (circle) was chosen as the final model, due to stagnation in validation loss.

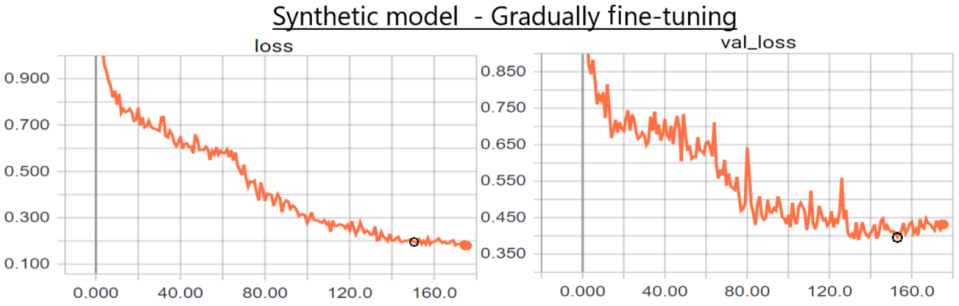


Figure 5.6: Training of gradually fine-tuned synthetic model over time. Left shows training loss, right shows validation loss. Epochs (layers, epoch size): 1 - 13 (heads, 1031), 14 - 64 (heads, 515), 65 - 129 (4+, 515), 130-176 (all, 515). Model after epoch 152 (circle) was chosen as the final model, at a point some epochs before loss increases.

5.2.9 Domain adaptation by fine-tuning and gradually fine-tuning

It was at the beginning of the experiment decided to use the synthetic model that performed achieved the highest mAP on the synthetic dataset due to a restriction in time.

We will in Section 5.2.11 see how the models were tested, but for now it is sufficient to say that that it was the gradually fine-tuned model that performed best on the synthetic test set, and therefore was chosen to serve as the target model from the real-world domain adaptation.

Domain adaptation was then done twice with the two different fine-tuning approaches, according to experiment overview of Section 5.1.2, and training setup and configuration of sections 5.2.6 and 5.2.7. As mentioned, there was a challenge and a drawback that there was no validation set to be used, due to the limited real-world training and testing data of 151 and 45 images at first, respectively. It was deemed a possibility to use some of the training images as validation data at the cost of less data for training, but it was

expected that the statistical significance would have been insufficient due to the resulting low number of images and number of vessels. Therefore, it was determined to use the first 45 images of testing dataset as the validation set. This was justified on the basis that no hyper-parameters was to be determined, that the real-world data augmentation scheme was already decided upon, and that only the number of epochs was to be determined best on validation scores. Moreover, since the numbers of epochs were to be determined by early stopping both for the adapted and real-world control models, the ground rules would be the same. On the other hand, if the experiment had aimed to find an optimal augmentation scheme, or to obtain an absolute measure of model performance for comparison with other previously trained and may be deployed model, the issue should have been resolved by having a distinct validation-set and test set.

It should then be noted that choosing the target model based on performance on the real-world test set could have been better since it can intuitively be thought to be the best basis for domain adaptation. However, at the point of determining the target model, this was thought to be invalid use of the test set since it would give the adapted models an unfair advantage, based on the discussion above. To sum up, this demonstrates the challenges that limited data pose, which in this case led to using the synthetic data for choosing the target model as a basis for domain adaptation.

The training progress over time is shown in figures 5.7 and 5.8. In addition to the combined validation loss, the bounding box loss, the classification loss, and the mask validation loss are also shown. When deciding which models to use, a compromise had to be made between optimizing the different losses. Classification loss was weighted the most, followed by the mask loss. This was also done for real-world control models later on as well. In retrospect, the model should have been based on an earlier epoch, considering that overfitting seems to have become present at epoch 151. However, model-weights before 151 was not saved for later use, and it was therefore not possible afterwards.

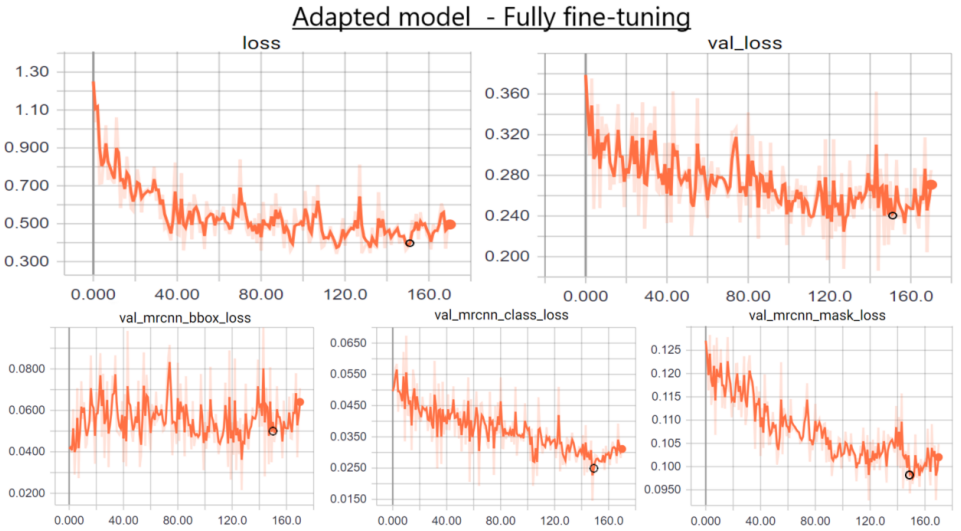


Figure 5.7: Training of fully fine-tuned adapted model over time. Epochs (epoch size): 1 - 149 (all). Model after epoch 149 (circle) was chosen as the final model, before bbox, class, and mask loss started to rise.

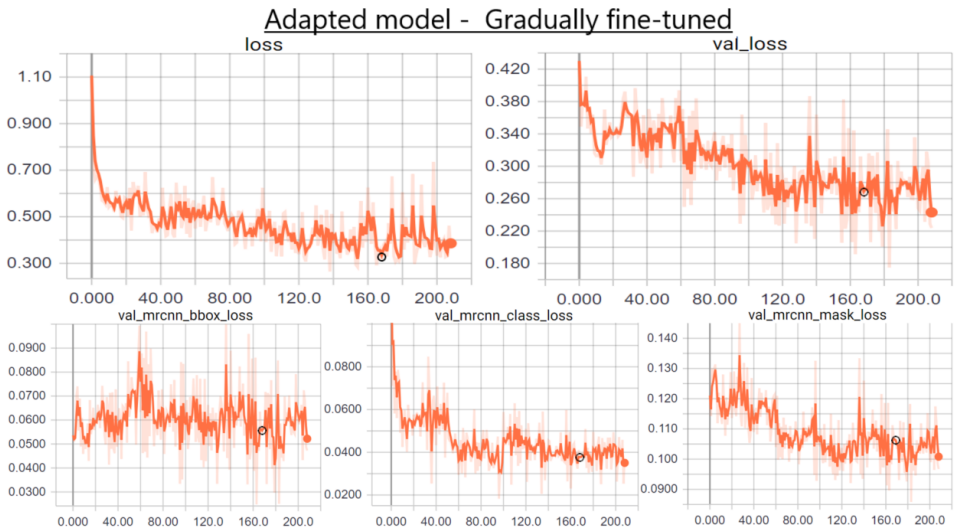


Figure 5.8: Training of gradually fine-tuned adapted model over time. Epochs (layers, epoch size): 1 - 48 (heads, 151), 49 - 88 (4+, 151), 89 - 208: (all, 151). Model after epoch 165 (circle) was chosen as the final model, due to a stagnation in loss.

5.2.10 Training of real-world control models

The real-world control models were trained in the same way as the adapted models, except that the target model used was the pre-trained Mask R-CNN model trained on the MS COCO dataset, which is the same target model as used for the synthetic models.

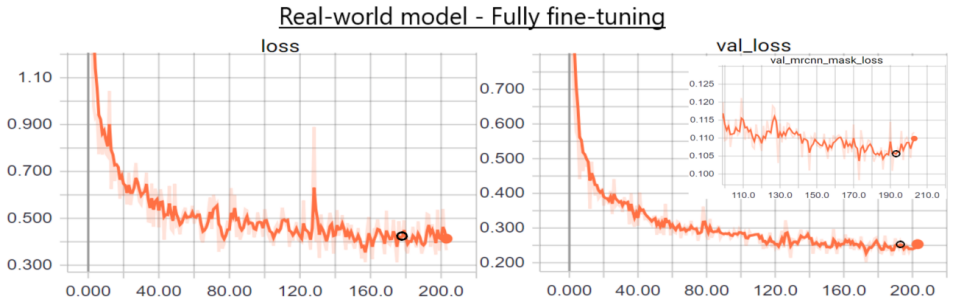


Figure 5.9: Training of fully fine-tuned real-world model over time. Epochs (epoch size): 1 - 203 (all). Model after epoch 193 (circle) was chosen as the final model due to stagnation in loss and signs of overfitting.

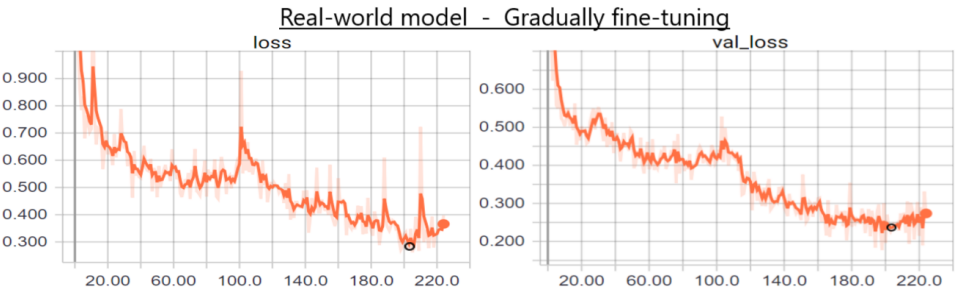


Figure 5.10: Training of gradually fine-tuned real-world model over time. Epochs (layers, epoch size): 1 - 48 (heads, 151), 49 - 88 (4+, 151), 89 - 125: (all, 151). Model after epoch 203 (circle) was chosen as the final model, before overfitting became present as observed by significant increasing validation loss.

5.2.11 Model testing setup

The common way of measuring and comparing object detection performance is to calculate and compare the mAP score (see Section 2.4.2). In addition, the precision-recall curves and the detection confusion matrix can give useful insight into the strength and weaknesses of the model detection performance. These three performance measures is used in the next chapter to test and compare models.

A script was therefore written to calculate mAP by:

1. Iterating through every image in a given test set, performs detection, keeping predictions with a confidence score higher than a given limit, correcting the remaining predictions as either true -or false-positive based on mask IoU higher than a given limit and correct class label, and saving these stats along with their confidence score.

2. Calculating the precisions and recalls for each class, and therefore indirectly constructing the precision-recall curve by the pairwise correspondence between the two sequences.

3. Calculating the area under each of the precision-recall curves using "all point interpolation".

4. Calculating the means of the areas under the curves.

Based on the precision-recall curves and true -and false-positive detections of actual objects, a precision-recall curve plot and a confusion matrix were also created.

It was observed that objects occasionally would be detected more than once with different detection labels, as shown in Figure 5.11. Therefore, it was decided to calculate scores based on allowing only one detection per vessel by keeping the detection with the highest confidence among duplicates. Two detections were considered duplicates if the IoU between their masks was greater than or equal to a certain threshold. This threshold was unpretentiously set to 0.9, based on the belief that no detection would be wrongly rejected with a limit that high. If a model should have been used in an application of some sort, this limit should admittedly be based on testing. This sort of filtering of detections was embedded in the procedure given by the list above.

Figure 5.11: Overlapping detections of a real-world cargo ship. The detection with label "Tanker" would be removed since it has the lowest detection confidence.

5.2.12 Explanation by LIME

It is of interest to see if we can gain some insight into what the instance segmentation models considers important for obtaining a object detection. Section 3.2.1 proposed a way to extend the model-agnostic method called LIME (see section 3.2) to give explanations

for masks as well as detection confidence, by scoring the mask quality based on the original mask prediction. To investigate this proposal, an experiment was conducted based on dedicating LIME to instance segmentation. Multiple methods has been attempted, and we shall take a look at two successful cases based on their respective experiments. We first present the dedication of LIME to instance segmentation, then the motivation and the setup for the two sub-experiments in connection with the methods. The gradually fine-tuned adapted model has been used for obtaining detections and mask predictions that was to be explained. The work has been based on an open-source implementation of LIME released by its authors [43]. Additional work were needed in order to dedicate this implementation to the chosen scoring scheme for detections. Segmentation algorithms used during the experiment are based implementations provided by scikit-image[54]. Note, the experiment based on their methods have been part of a trial-and-error phase in order to obtain a working solutions. A selection of the resulting explanations that gives insight into the quality of the explanations are presented in along with the other results in the next chapter.

Dedication of LIME to instance segmentation

We start by considering the segmentation of images, as this is a prerequisite for obtaining explanation for any image processing task with LIME. The aim is according to [43] to segment the image and objects into semantically meaningful segments or super-pixels which we humans consider as important characteristics which should be included in an explanation, as to assert trust in the model at the locality of the image. Figure 5.12 illustrates different segmentation algorithms and segmentation results with different degree of specificity, implemented using the scikit-image[54] software library. Whether it is best to apply specific and dedicated segmentation for each object or to apply a general segmentation scheme is uncertain. A specific segmentation such as Felzenszwalb's segmentation algorithm as seen in the figure may become biased towards the object and prevent the surroundings from being taken into consideration, while a to general segmentation may result in wasting time on unimportant aspects. We have not stuck with a specific segmentation scheme or algorithm is this experiment, but rather experimented using different ones. In the second part of this two-stage experiment we use two different segmentation algorithm. However, the task of deciding whats good practice is left for future work. Furthermore, the segments that are to be replayed during perturbation of an image are set to be replaced with the mean value of all pixels in the image not part of the original predicted masks of objects, as to introduce as little bias as possible. Figure 5.13 illustrates detections based on a original image and perturbations based on pre-segmentation.

Next, we consider how to score a detection based on a segmented image, that is how to score the detection and classification of an object and how to score the predicted mask. We start by omitting those detection that has an IoU between the predicted mask and the original predicted mask less than a certain threshold. We set this threshold to be 2.5 %, such that even relative small masks are accepted, while those that are totally off the object are omitted. It is unlikely that such omitted detections are part of the explanation, and thus it is reasonable to omit them. The score of a detection can simply be taken as the

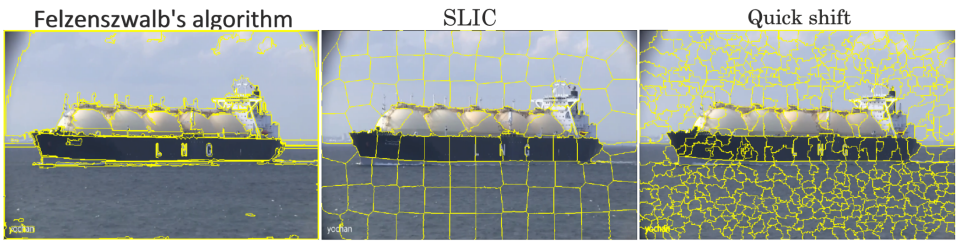


Figure 5.12: Illustration of different segmentation algorithms. Felzenszwalb's algorithms obtains a very specific and dedicated segmentation, the SLIC algorithm obtains a coarse and general segmentation, and Quick shift obtains a fine and general segmentation. The segmentation result is dependent on the hyper-parameters of each segmentation algorithm. Implemented

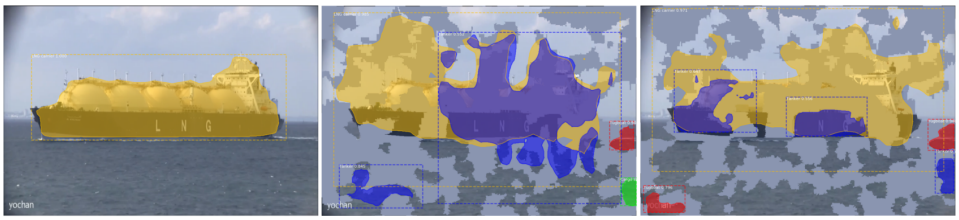


Figure 5.13: Illustration of detection from an original image and perturbed versions based on segmentation.

detection confidence, already provided by the detection. The score of the mask quality can be based on the IoU between the predicted mask and the original predicted mask from the original detection that is to be explained. IoU has the advantage that masks which is inside the original mask gets penalized less than masks that have the same size but which are partly localized outside the original mask. However, the drawback is that this way of scoring predicted masks ignores the fact that masks which is fully contained in the original mask may be as optimal as one could expect given a specific set of segments. For instance, if a considerable side of a vessel is completely hidden while the rest of the vessel is completely visible, the optimal and intuitively correct mask would be the one that covers the visible part. Thus, the score for such a case will be unjustifiably low. We leave the solution for a metric that takes this aspect into consideration as further work on the topic. Nevertheless, it will be interesting to see how these two types of scoring schemes will affect the explanations. Finally, we expand the scores for an object to include all possible classes. Scores for a specific predicted class is therefore assigned to that class. This expansion is useful, since it allows us the also gain insight into why a model would consider one type of vessel to be another type of vessel. The result was that each detected object gave eight different scores, two for each possible class.

The LIME implementation requires a function pointer to a scoring function, that takes a set of images as input, and that returns a two-dimensional array containing the scores. The number of rows corresponds to the number of perturbed images based on the same image, while the columns correspond to the number of explanations that are to be obtained

from the original image. Therefore, a scoring function was written based on access to the original detections. The original detections were used for determining the size of the output array based on the number of detections, such that a given number of original detections gave an output array of length equal to eight times the given number of original detections. Since the number of original detections is constant, the length of the output array is also constant. In addition, the original mask predictions were used to calculate the mask scores.

Two objects experiment

The first part in this two-stage experiment involves an image with two vessels, one cargo ship and one tugboat. There will be two sets of pairs of explanations, one set of pairs for the cargo ship and one set of pairs for the tugboat. Detection confidence explanations and mask IoU explanations constitute the pair.

The primary aim is to see whether the method is able to obtain two explanations that is clearly separate, or if there is a relationship between the explanations. The secondary aim is to see whether the explanations assert trust in the sense that important characteristics of the vessels have been taken into consideration. Since the tugboat is small in size compared to the cargo ship, it is believed that the explanations will be coarse. On the contrary, the cargo ship is large in size compared to the image, and therefore it is believed that the explanation is finer and will highlight important characteristics. The image and the original detection along with the segmented image are shown in Figure 5.14. The third and last aim is to see if there is a difference within the pair of explanations, that is if there is a difference between the detection confidence explanation and the mask explanation.

The image is segmented into 150 segments using the watershed algorithm, and the number of samples are set to 2000. The watershed algorithm treats the gray-scale variant of the image as a topological map with height based on the pixel brightness, and finds the lines that run along the tops of ridges. It is important that the number of samples exceeds the number of segments, such that there is a significant statistical basis for the regression. It took approximately 326 minutes to obtain the explanations. In short, a considerable amount of time. The explanations are presented in along with the other results in Section 6.5.1.

Single object experiment

The second part in this two-stage experiment involves four images with a single vessel, each of one class. For each image, we consider a set of explanations for all vessel-classes. The first aim is to see which characteristics of a ship belonging to a particular class that are considered essential for a detection's particular explanation, and for obtaining false-positives detections for the other vessel classes. For instance, this may give insight to why a LNG carrier may be wrongly detected as a tanker.

First, all images are segmented into between 80 and 100 segments using the SLIC algorithm and the LAB color space as recommended by scikit-learn. SLIC uses k-means clus-

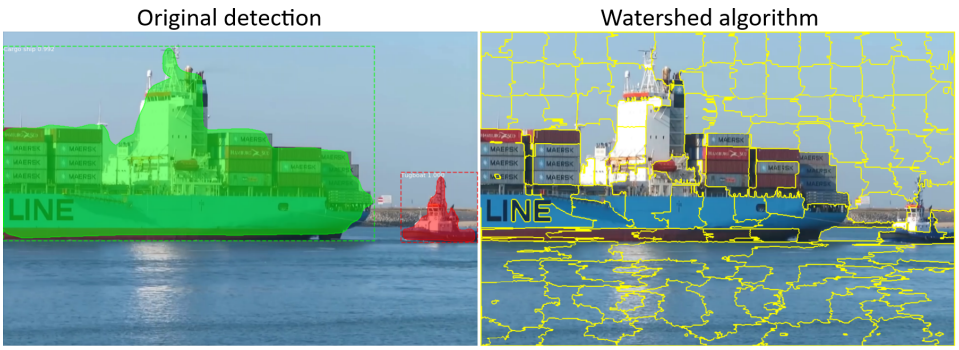


Figure 5.14: Original detection and super-pixel segmentation of image with a cargo ship and a tugboat in first LIME experiment. Left-hand side: The original detections. Right-hand side: Segmented image with 150 segments by using the watershed algorithm.

tering in the chosen color space of the image. The hyper-parameters have been adjusted to obtain a reasonable segmentation for each image. The original detection of objects, the predictions of masks, and the segmentation of the images are shown in Figure 5.14. Then,

Figure 5.15: Original detection and super-pixel segmentation of images in the second LIME experiment. LNG carrier: 88 segments. Cargo ship: 100 segments. Tanker: 92 segments. Tugboat: 104 segments.

LIME and Google Colaboratory was used to obtain four sets of explanations for each of the detections, one for each class. The number of samples was set proportionally to the number of segments in each image by a factor of ten, which was found to give a sufficient statistical foundation for the regression, and by such providing clear explanations. Run-time per detection was on average 10.03 seconds, such that the total run-time became considerable large. An overview is given below:

- LNG carrier: 88 segments, 880 samples, 2 hours and 26 minutes.
- Cargo ship: 100 segments, 1000 samples, 2 hours and 46 minutes.
- Tanker: 92 segments, 920 samples, 2 hours and 35 minutes
- Tugboat: 104 segments, 1040 samples, 2 hours and 55 minutes

Chapter 6

Results

This chapter first presents performance results from all models trained in the previous chapter. The models were tested on both the synthetic and the real-world test set to assess their performance across the domains. In addition, the models that constitute each step during gradual fine-tuning have been tested to see if they can give additional insight into the workings of domain adaptation. Second, this chapter presents the results from the LIME experiments. Visual explanations for different detections are presented. Interpretations and observations are commented while results are presented, and further discussed in the next chapter.

6.1 Performance results

All performance scores of the different models are shown in Table 6.1 and visualized in Figure 6.1. Scores are given as mAP@[50:70] and mAP@[75:90] in percentage, that is the average mAP over different IoU thresholds. These two metrics give a broader and clearer view of the performance than what isolated mAP values can give. mAP@[50:70] can be interpreted as a measure of how good the models are to detect objects in images, while mAP@[75:90] can be interpreted the same way but with a stricter requirement for mask accuracy and thus how good the models are on both detecting and locating the objects. These two metrics are suitable for comparing models. However, average mAP does not give sufficient insight into the precision and recall for each model, nor does it give insight into the performance at each class. Therefore, the precision-recall curves and confusion matrix are also provided in the next chapter, when seen as necessary to gain sufficient insight. For instance, the precision-recall curve will give insight into why the scores for each metrics was approximately at best 92 mAP@[50:70] and 68 mAP@[75:90].

Performance measured by average mAP[%:%].
(average mean Average Precision over different IoU thresholds.)

	Synthetic testset		Real-world testset	
	mAP@[50:70]	mAP@[75:90]	mAP@[50:70]	mAP@[75:90]
Synthetic models				
Fully fine-tuning	85,44	31,18	58,37	33,44
Grad. fine-tuning, heads	86,92	27,07	73,97	27,33
Grad. fine-tuning, 4+	90,61	**37,69**	69,01	37,84
Grad. fine-tuning, all	**90,63**	37,35	75,77	43,10
R-w control models				
Fully fine-tuning	39,83	7,31	85,20	55,48
Grad. fine-tuning, heads	52,47	6,42	88,40	54,66
Grad. fine-tuning, 4+	51,47	13,44	87,15	64,55
Grad. fine-tuning, all	63,94	15,19	91,22	67,74
R-w adapted models				
Fully fine-tuning	86,64	36,83	90,41	66,41
Grad. fine-tuning, heads	84,59	30,53	**92,78**	60,27
Grad. fine-tuning, 4+	89.00	32,07	90,70	65,98
Grad. fine-tuning, all	87,88	33,96	91,56	**68,60**

Table 6.1: Results from all Mask R-CNN models. Only the most confident detection among overlapping predicted masks with an IoU greater or equal to 90 % has been kept. Numbers represents performance scores in percentage based on allowing no duplicate predictions (see Section 5.2.11). Best performance per metric overall is highlighted in **bold**, while best performance per metric within the different groups of models are highlighted with underline.

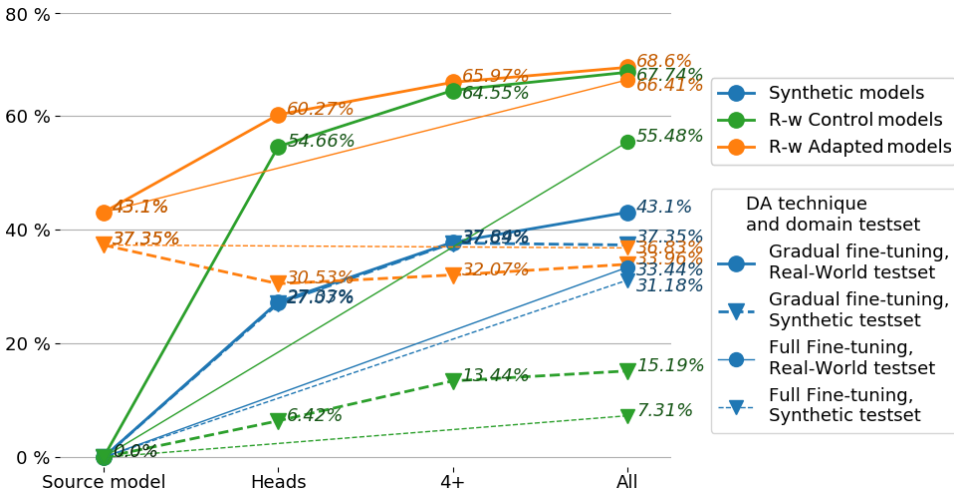


Figure 6.1: Visualized results from all Mask R-CNN models at mAP@[75:90]. Upper legend describes the color encodings based on category of models, while the lower left legend describes the line -and marker-styles encodings based on the different domain adaptation techniques and the two test sets from their respective domains.

6.2 Synthetic models performance

Performances of the synthetic models on the synthetic test set (see *Synthetic models* in Table 6.1) are persistently high and quite similar when considering mAP@[50:70], while they drop when considering mAP@[75:90] with more than 50 pp between the best scores. These results suggest that it is easy to correctly detect and locate the different types of vessels in the synthetic dataset, while it is harder to obtain accurate masks. Best performance at mAP@[50:70] is achieved by the gradual fine-tuning all-layers model with a score of 90.63 %, while the gradual fine-tuning 4+ model achieves the best performance at mAP@[75:90] with a score of 37.69 %. The performance differences between these two models are small, suggesting that performance improvement has stagnated at the second step of the gradual fine-tuning approach. Worst performances are achieved by the gradual fine-tuning of overheads and full fine-tuning with a noticeable distance in scores compared to the other two fine-tuning approaches. A visual interpretation of the performance on the synthetic test set is seen in Figure 6.2.

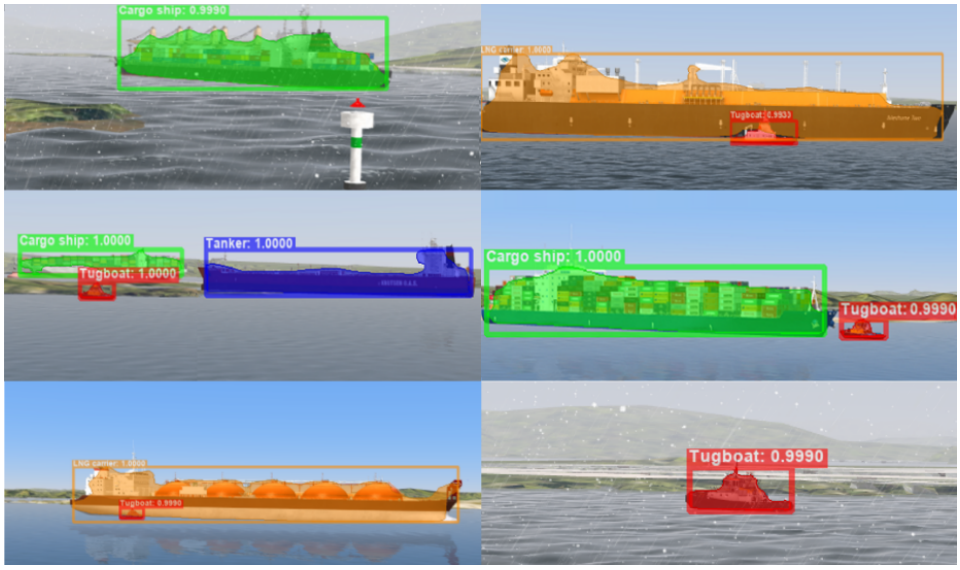


Figure 6.2: Gradually fine-tuned synthetic model all-layers performing on the synthetic testset. Images has been cropped around the bounding boxes to make detections more visible.

Moreover, performance on the real-world data drops when considering mAP@[50:70], but actually persists and even increases when considering mAP@[75:90]. Gradual fine-tuning toward all layers achieves the best performance of 75.77 mAP@[50:70] and 43.10 mAP@[75:90], were the former score should be considered high. Since it is expected that the covariate shift would make it challenging for models trained purely in one domain to perform in another domain that it has newer learned from, it is unexpected that performances would persist and even increase. This is likely due to what was earlier described as a sampling selection bias (see Section 5.2.3), being that the domain-invariant features such

as range, orientation, and combination of objects and coastal environment are distributed differently. In short, the real-world test set is likely less challenging than the synthetic test set. Therefore, it is plausible that mAP@[50:70] drops closer to mAP@[75:90] because challenging detection tasks in the synthetic testset does not appear in the real-world testset, while the simpler does. A critical observation is that gradual fine-tuning experiences an significant increased performance of 4.07 pp at mAP@[50:70] and 5.26 pp at mAP@[75:90] on the real-world test set after the performance improvement has stagnated on the synthetic test set. Thus, knowledge about domain-invariant features continuous to increase despite stagnation on the synthetic test set. This observation is crucial, since it tells us that this improvement is not observable in the synthetic domain, and that we cannot tell if this progression would have continued to increase, and if so for how much, had the training lasted for longer. We discuss this observation in the next chapter.

Since the instance segmentation task involves multiple classes, it is also interesting to take a look at the performance of the models with respect to the different classes. We base our self on the best performing adapted model, that is the gradually fine-tuned all-layers model. The precision-recall curves at mAP@50 and mAP@75 and the confusion matrix at mAP@75 are shown in Figure 6.3 and gives insight into the detection performance of the model among the different vessel classes. The precision-recall curves are for mAP at single IoU values only, since precision-recall curves for average mAP are not defined. It was suggested that the drop in score from mAP@[50:70] to mAP@[75:90] was because it is easy to correctly detect the different types of vessels while it is harder to obtain accurate masks. The precision-recall curves confirm this claims as the recall is close to one for all the vessel classes at mAP@50, but drops dramatically for mAP@75. It is also evident that it is the tugboat class that contributes the most towards a decrease in mAP@75, followed by the tanker class and then the cargo ship class. From the precision-recall curve, it is seen that the model detects six false-positive tugboats after the first seven true-positives, sorted by decreasing detection confidence score. This results in that the scores drop significantly before the next stream of true-positives. Then there are an additional seven false-positives. If the false-positives at the start have had lower detection confidence, the score from the tugboat class would have been notably higher. This is partly true for the tanker class as well, but it does a better job in determining the detection confidence. From the confusion matrix based on mAP@75, it is seen that model only detects 33 out of the 58 tugboats, comparably lower than for the other vessel classes. In addition, the model wrongly detects four cargo ships and one LNG carrier as a tanker, and one LNG carrier as a cargo ship. There is a good reason to believe that class imbalance is partly responsible for these results. We will discuss this further in the next chapter.

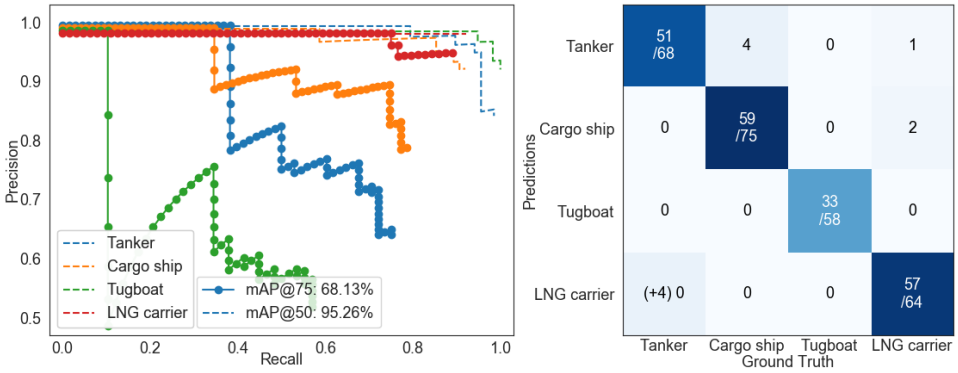


Figure 6.3: (Left) Precision-Recall curves at mAP@50 and mAP@75 and (right) confusion matrix at mAP@75 for the gradually fine-tuned all-layers synthetic model performance on the synthetic test set. The curves have slightly been vertically shifted to highlight the details. Bottom integer numbers on the diagonal of the confusion matrix are the total number of class instances. Duplicate and omitted detections (see Section 5.2.11) are shown in the parenthesis.

As a final note, the performance of the synthetic models on the synthetic test set has likely been hurt by the data augmentation scheme used during training. The augmentation was intended to increase the adaptation ability of the models towards the real-world domain by increasing the diversity of the synthetic training-set in terms of colors, shadows, edges, and occlusion. This has likely made the synthetic models under-fitted for synthetic data. Moreover, augmentation can explain why the all-layers gradually fine-tuned model performs best on the real-world testset, while only second best on the synthetic test set. It can simply be that since the synthetic models are trained on augmented synthetic data, the difference in performance becomes visible only when the models are challenged on difficult detection tasks, while is non-visible when they are challenged with simple non-augmented synthetic images.

6.3 Real-world control models performance

Performance of the pure real-world control models on the real-world test set is impressively high, considering that the size of the training-set is only 151 images and between 69 and 82 vessels per vessel class. This may indicate that the real-world training-set has been sufficiently diverse as for a model to be able to learn features corresponding to different viewpoints and across different vessels within vessel classes. Best performance is achieved by the gradually fine-tuned all-layers model which achieves 91.22 mAP@[50:70] and 67.74 mAP@[75:90], while the worst performance is achieved by fully fine-tuning which achieves 85.20 mAP@[50:70] and 55.48 mAP@[75:90].

Despite this, performance drops significantly in the synthetic domain with the best scores being 63.94 mAP@[50:70] and only 15.19 mAP@[75:90]. On average, the drop in performance is 36.06 at mAP@[50:70] and 50.02 percentage-points at mAP@[75:90]. Since

the best mAP@[50:70] score are that low, it is justified to assume that the control models are not able to detect a significant portion of the vessels. Besides, the control models could be considered incapable of segmenting vessels with high accuracy in the synthetic domain since the best mAP@[75:90] score is that terrible. This stands in contrast to the synthetic models, where the performance at mAP@75 persisted between the domains and supports the claim that the synthetic test set is more challenging.

6.4 Real-world adapted models performance

When comparing the performance of the real-world adapted models with the performance of the synthetic source model on the real-world dataset, that is the synthetic gradually fine-tuned all-layers model; it is clear that domain adaptation has occurred. All adapted models achieve a significantly better score than the source model. Overall, the performance of all the real-world adapted models on the real-world test set is persistently high at mAP@[50:70], and relatively high but more spread at mAP@[75:90]. Moreover, the adapted models persistently achieve the best scores on the real-world test set compared to the corresponding control models. The larges difference between adapted models and the corresponding control models is between the fully fine-tuned ones, where the adapted one achieves a 10.03 pp higher mAP@[75:90] score and a 5.21 pp higher mAP@[50:70] score. The smallest difference is between the gradually fine-tuned all-layers models, were the adapted one achieves a slightly better score in both mAP@[50:70] and mAP@[75:90]. Best performance among all models on the real-world test set when valuing mask accuracy that is high mAP@[75:90] score, can be considered achieved by the adapted and gradually fine-tuned model; it achieves only 1.22 pp less than the best performing model at mAP@[50:70], that is the gradually fine-tuned overheads only model, while 8.33 pp more than the same model at mAP@[75:90] with a score of 68.60 mAP@[75:90]. Second best performance can be considered achieved by the adapted and fully fine-tuned model; it achieves 2.37 pp less than the best performing model at mAP@[50:70], while 6.14 pp more than the same model at mAP@[75:90] with a score of 68.60 mAP@[75:90]. In fact, the best performing model at mAP@[50:70] achieves the worst score among the adapted models at mAP@[75:90], which means that it is most capable of detecting object while worst in prediction accurate masks of detected objects.

When considering testing on the synthetic test set, performance remains consistently high at mAP@[50:70] for all adapted models with a 7.09 average percentage-points drop from the real-world test set to the synthetic one, that is when averaging over the differences. Likewise, there is a 31.97 average percentage-points drop at mAP@[75:90]. Despite the latter seems dramatic, it is not that bad when comparing the adapted models to the control models. Rather, the results are impressive as the scores are overall very similar to that of the pure synthetic models. Compared to the source model, that is the gradually fine-tuned all-layers synthetic model, the average percentage drop is only 3.61 at mAP@[50:70] and 4.00 at mAP@[75:90]. In addition, the fully fine-tuned model experience only a 0.51 percentage-point drop and the gradually fine-tuned all-layers model experience a 3.39 percentage-point drop at mAP@[75:90]. Besides, gradual fine-tuning experiences an increase the performance in the two last steps after a drop in the first step as seen in the

visualized results. This is staggering because it suggests that the adapted models have been able to maintain knowledge about the synthetic domain and avoids any strong signs of catastrophic interference (see Section 2.5.2). Furthermore, this suggests that the adapted models is in possession of a far larger portion of domain-invariant features than the control models.

A visual interpretation of the performance can be seen in Figure 6.4. The masks, bounding boxes, and vessel labels can be considered accurate. Two false-positive detections can be observed, where non-vessel elements are falsely detected as vessels. A breakwater with a lighthouse is falsely detected and classified as an LNG carrier in the above image, while the shore is falsely detected and classified as a tanker in the lower image. By imagination, it is understandable that this may occur, as features in the elements may be found in LNG carriers and tankers. A shore with odd shapes and variations in colors in the distance can be interpreted as a ship even by humans.

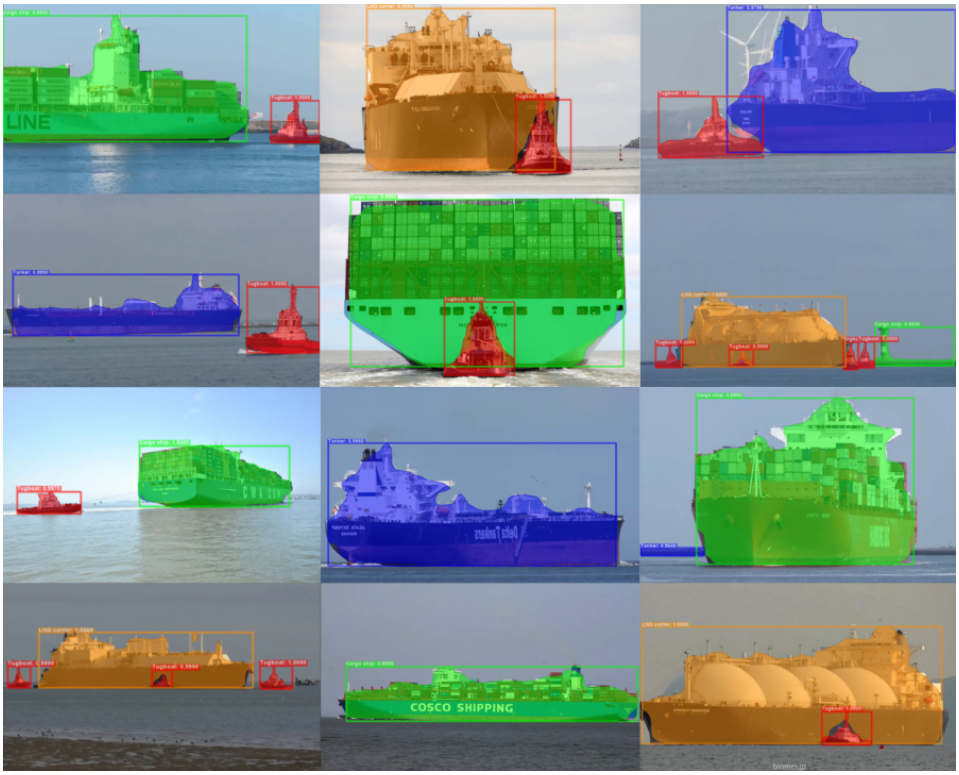
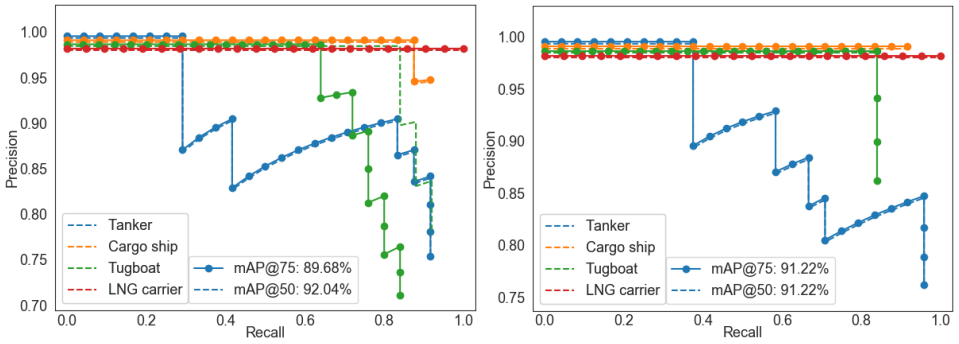


Figure 6.4: Gradually fine-tuned all-layers adapted model performing on the real-world test set. Images have been slightly cropped around bounding boxes to make detections and tags more visible despite re-scaling of images. Two false-positive detections can be observed in the second and third row of the last columns, where a breakwater with a lighthouse and shores are falsely detected as an LNG carrier and a tanker with detection confidence scores of 0.883 and 0.964 respectively.

We shall gain more insight into the workings of the models by looking at the precision-

recall curves shown in Figure 6.5. As a supplement, the precision-recall curves for both the best control model and adapted model have been included. It is seen that it is false-positive tanker-detections that contribute most towards loss in both mAP@50 and mAP@75 for both models. The control models achieve a slightly higher recall-value than the adapted models while assigning higher detection confidence to the false-positives, which results in a higher loss regarding the tanker class. Next follows the tugboat class. Both models experience a loss due to achieving a recall value is slightly above 0.8. However, while the control model has only four false-positives and assigns them the lowest detection confidence scores, the adapted model have as much as eight false-positives and assigns as much as six of them a higher confidence score than one or more true-positives. Therefore, the adapted models perform worse than the control model in regards to the tugboat class. Furthermore, the models experience a minor loss due to the cargo ship class. In comparison, this is a similar pattern that was observed for the synthetic gradually fine-tune all-layers model as well. There, the tugboats contributed the most towards a decrease in loss followed by the tankers and then the cargo ships. While both models struggle with the tanker class, the difference in performance with regards to the tugboat class is noticeable, suggesting that the adapted model has a bias of some sort that makes it prone to detecting false-positive tugboats. We discuss this in the following chapter.



(a) Precision-recall curves for gradually fine-tuned adapted model

(b) Precision-recall curves for gradually fine-tuned control model

Figure 6.5: Precision-recall curves for the best performing model among the adapted models and the control models, being the gradually fine-tuned models. The curves have slightly been vertically shifted to highlight the details. False-positive tanker-detections contribute the most towards a decrease in loss followed by false-positive tugboat-detections for both models.

6.5 Model explanation by LIME

We have conducted an experiment with LIME based on two images, one image with two objects, and one image with a single object. The results are exclusively visual, and we present multiple sets of images to present and to gain insight.

6.5.1 Two objects experiment

The results are shown in figures 6.6 and 6.7, with different number of segments or features. Firstly, it is clear that the two type of explanations successfully values segments in their respective object in the image. For instance, the explanation for the cargo ship is based on segments from the cargo ship. Secondly, the explanation for the cargo ship gives us some insight into what characteristics of the vessel the model has considered important. Based on the increasing number of segments in the explanations, it seems like the waterline and hull below the bridge, and the hull at the stern are most important, while the containers and the bridge comparably are less critical. However, the explanations do not suggest whether the most values segments are important for predicting the presence of a vessel in general or a cargo ship in specific. It could be that the waterline and hull below the bridge along with the stern is this case is important for determining the presence of a ship, while the containers are important for classifying the ship. On the contrary, the highest valued segments in the explanation for the tugboat is too coarse and suggest no more than the location, while the less valued segments suggest that the waterline below the hull of the container ships are important. There could perhaps be that the model has learned that the presence of another object likely means that the smaller object is a tugboat. This explanation makes sense since tugboats are often seen in proximity to other vessels in the dataset. While this may be the case, it could also be that the number of segments included in the explanation exceeds the reasonable numbers in terms of valued number of segments, at that the last segments are the least worse among all remaining ones. Thirdly and lastly, there are also some differences between the two types of explanations both for the cargo ship and the tugboat. The mask prediction explanation for the container ship seems to value the containers and bridge to a higher degree than the detection confidence explanation, and the first segment in the tugboat explanations disagrees on whether it is the bridge or the hull that is the most important characteristics.

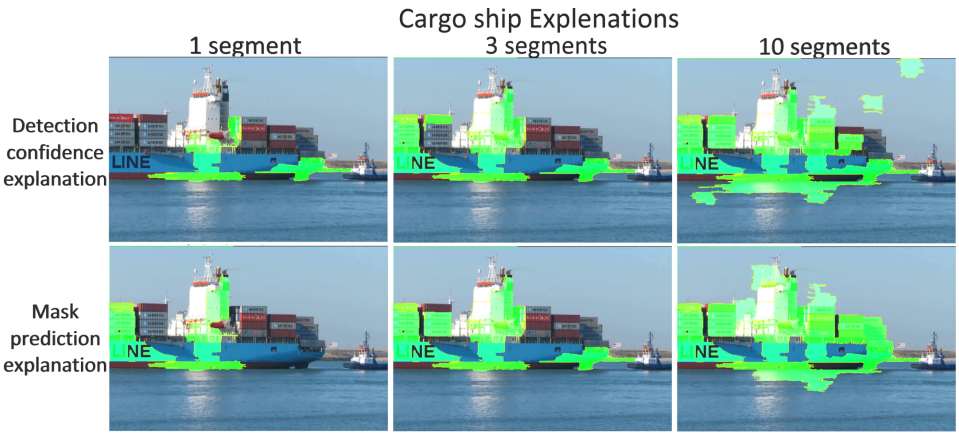


Figure 6.6: Explanations for the cargo ship in the first LIME experiment. The segments explains why (green) the given cargo ship may be detected as cargo ship. The segments in the image on the left are valued (highest weights) the most, followed by the extra segments in the next images towards the right.

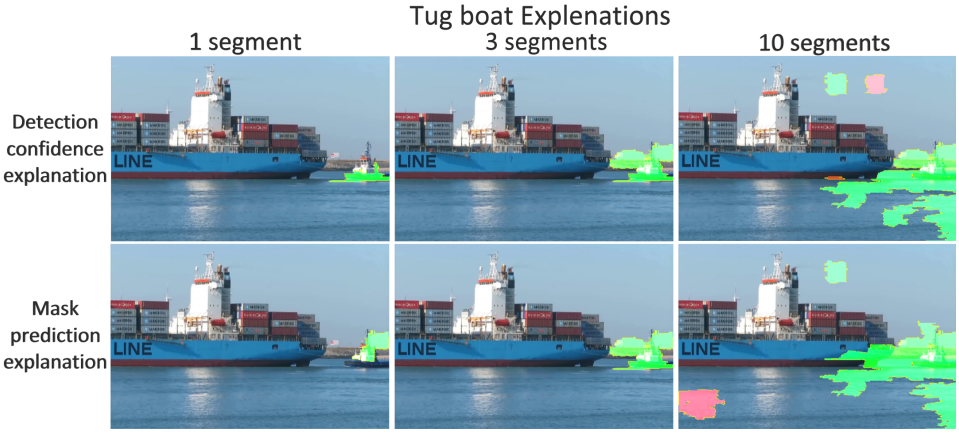


Figure 6.7: Explanations for the tugboat in the first LIME experiment. The segments explains why (green) or why not (red) the given tugboat may be detected as tugboat. The segments in the image on the left are valued (highest weights) the most, followed by the extra segments in the next images towards the right.

6.5.2 Single object experiment

We take a look at a selection of explanations given by the experiment, which gives insight into the quality of these explanations.

Firstly, Figure 6.8 shows explanations for why a specific cargo ship and which parts of the cargo would result in high detection confidence for a specific class, based on the 20 highest weighed segments out of a total of 100 segments. We see that the containers on the ship

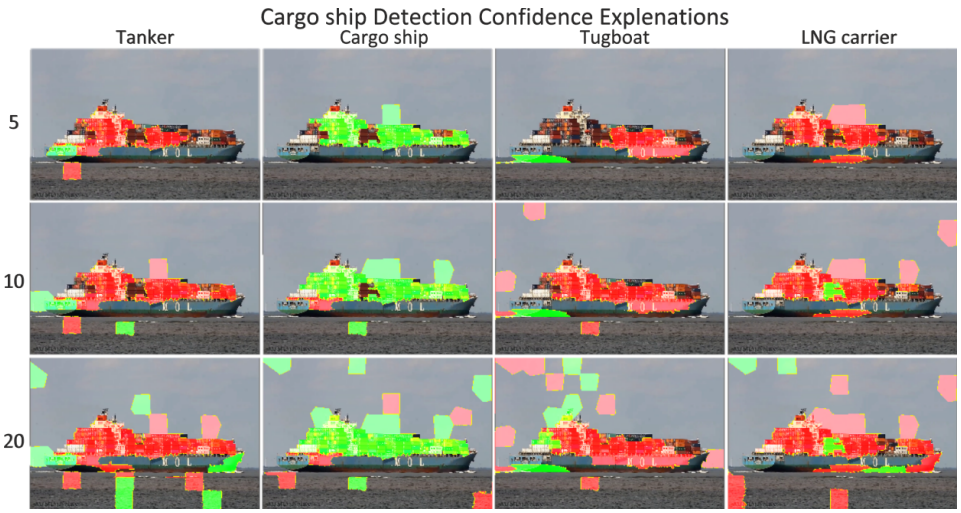


Figure 6.8: Detection confidence explanations for a cargo ship in the second LIME experiment. The number of segments valued (highest weights) the most are shown on the left-side of the images. The segments explains why (green) or why not (red) a cargo ship may be detected as a certain class given by the labels above the rows.

seem to be most important for detecting the cargo ship and for discriminating between the different vessels classes since the first five segments highlight them in green or red. We also see that the hull is less important for detecting the cargo ship, while it is important for not detecting a tugboat. Furthermore, we see that there are indications that the model has considered a collections of containers in the middle of the ship valuable for detection an LNG carrier, a collection of containers at the stern and the top of the bridge valuable for detecting a tugboat, and the bow and the stern valuable for detecting a tanker. This explanation asserts trust in that the model considered the containers valuable for detecting a container ship. However, the fact that it does only value the bridge to a moderate degree and the hull to a small degree does not assert trust as it suggests that the model would detect container ships as long as there is a collection of containers present. Nevertheless, this is reasonable as the model has only seen containers in connecting with cargo ships and shouldn't, for instance, be expected to be able to distinguish between elements in a port and a container ship.

Secondly, Figure 6.9 shows explanation the same type of explanations as Figure 6.8, but for all vessel classes and the ten highest weighted segments only. We see the same pattern in this figure as in the previous figure for all the vessels, being that the unique characteristic is considered most important for correct detection and classification while also being important for avoiding false-positive detections. For instance, we see that the mast at the bow and the surface on the deck of the tanker, and the liquid natural gas tanks on the LNG carrier, are both positively and negatively highly weighted across the explanations. Furthermore, we see that parts of the bridge of the tanker are highly valued for classifying the vessel both as a tugboat and as an LNG carrier, which suggests that this characteristic can

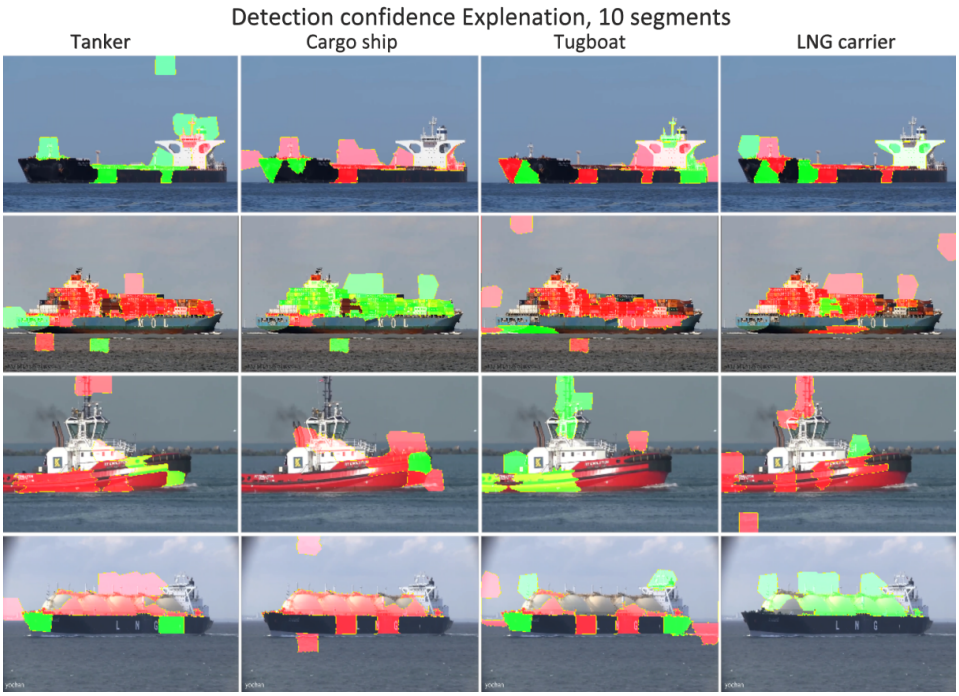


Figure 6.9: Detection confidence explanations for all vessel classes across all vessel classes. The segments explains why (green) or why not (red) a given vessel may be detected as a certain class given by the labels above the rows.

be a source to confusion.

Thirdly and lastly, Figure 6.9 shows explanations for which part of specific ships are important for predicting the original mask prediction (see Figure 5.15 for the respective mask predictions of the images), based on the 20 highest weighed segments. We see the same pattern in this figure as in the other figures for all the ships, being that the unique characteristic is considered important. However, this time the waterline and the hull of the cargo ship and the LNG carrier are considered more important than before. This is reasonable, as the boundaries of the object could be important for obtaining an accurate mask prediction. In addition, we see that the characteristic cone-shape of the tugboat is valued.

Considering all the figures, we see that the most valued segments of the tanker are smaller in size and less uniquely characteristic than those of the cargo ship and the LNG carrier. For instance, the bridge of the tanker is not valued at all by the ten most valued segments. We know from the precision-recall curves for the model (see Figure 6.5a) that the tanker vessel class is a source of high loss compared to the other two. These observations suggest that it is, in fact, due to the lack of unique characteristics.

Overall, these explanations assert trust in that the model values the unique characteristics of the vessels. However, as noted in connection with Figure 6.8, the model seems not to

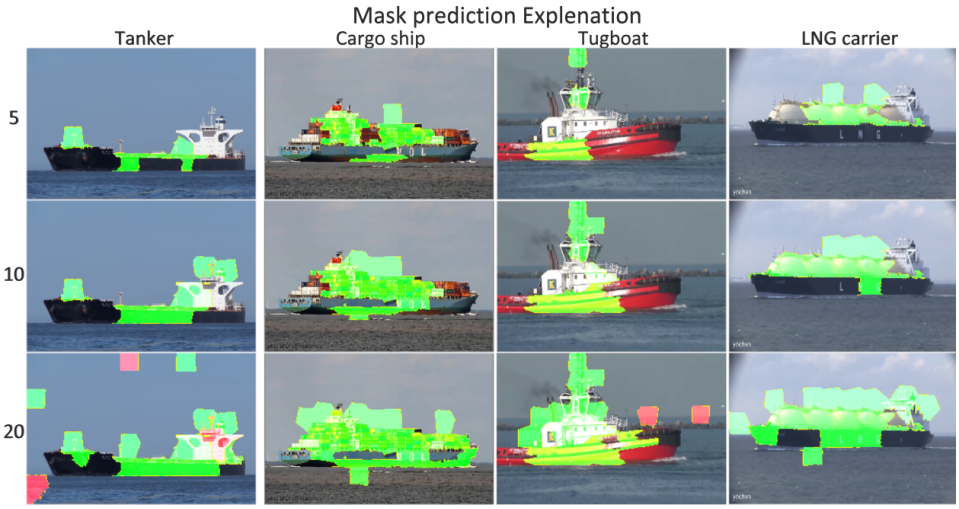


Figure 6.10: Mask prediction explanations for all vessel classes in the second LIME experiment. The number of segments valued (highest weights) the most are shown on the left-side of the images. The segments explains why (green) or why not (red) a given vessel may be detected as a certain class given by the labels above the rows.

value the combination of unique characteristic and general vessel characteristic. Alternatively, it is not visible in the explanation. The implications of this lack of relationship is discussed further in the next chapter.

Chapter 7

Discussion

7.1 Potential sources to error

7.1.1 Class imbalance

We have seen that there was a significant extrinsic imbalance between the vessel-classes in the synthetic training-set (see Table 5.1 in Section 5.1.2) in favour of the tanker vessel-class and disfavour of the tugboat vessel-class, due to the external factors during the acquisition of the dataset. We have also seen the best performing synthetic model overall performed worse on the tugboat vessel-class compared to the other vessel-classes (see precision-recall curves and confusion matrix in Figure 6.3 in Section 6.4), and that there was an indication that the best performing adapted models overall performed worse with respect to that class when comparing performance between the best performing adapted model and the best performing control model (see Figure 6.5 in Section 6.4). Therefore, it is good reason to believe that class imbalance has affected the performance of the synthetic source model and consequently, the adapted models. Class imbalance is known to have a detrimental effect on a deep learning models, affecting both convergences during the training phase and generalization during testing phase [55]. A likely explanation is that the tugboat vessel-class has lost the battle for model capacity in favour of the other classes during the training of the source model which has resulted in a lacking ability to generalize to that class. In other words, the tugboat vessel-class has likely obtained less model capacity compared to the obtained model capacity of the other classes. This lack of ability has then likely further been passed to the adapted model. There exist multiple methods for addressing the imbalance, but since the imbalance in our case is extrinsic, the best effort would be to balance the dataset during the acquisition phase. This likely impact on performance due to class balance shows that it can have detrimental across domain adaptation, since there was a class balance in the real-world data.

7.2 Synthetic data

7.2.1 Acquisition

We delayed a discussion regarding K-Sim and the developed acquisition method from Chapter 4 to this chapter. There, we saw how K-Sim could be used to produce synthetic images and covered in details the method for acquiring accurate instance-level label images given sufficient thresholding of the source images. Furthermore, the experiment showed how the acquisition method allowed for exploring the practical implication for using synthetic images produced by K-Sim. This ability is valuable, considering the unfamiliarity regarding synthetic data. However, there were room from improvement regarding thresholding (see Section 4.2.3). Also, we also see that the method does not allow for exploiting the full potential offered by K-Sim. The main reason for this was that there was no access to the internal working of the software such as the rendering process during the work, and such that an alternative approaches had to be devised instead, as described in Chapter 4. We summarize the limitations and the unexploited potential below:

- Real-time labeled images is not provided, since labeled images is separated by multiple seconds in best case scenario due to the reliance of sufficient synchronization
- Only deployable objects is labeled.
- Tracking information is not provided.
- Depth information is not provided.
- Information about states in the simulator is not provided.
- Scenario information is not provided, e.g. collision course, vessel interaction, and emergency signaling.

From the list above it is clear that we have only scraped the surface of K-Sim's potential. While information about the states of the simulator are visually available in the user interface, it would require a delicate acquisition process beyond what has currently been implemented. The other unexploited potentials is the list above demands another acquisition solution based on software access.

Furthermore, the acquisition process is relatively time-consuming relative what could be expected from using computer graphics to generate synthetic data. This is mainly due to the simulator running in real-time, which made image acquisition based on the screen capture process painfully slow. It total, it took 10 days to create approximately 1.200 images, while labeling only took below 9 hours. In comparison, a German team from University of Darmstadt [34] was able to extract 20.000 unique images with labeled with pixel-accuracy from a video game in less than 40 hours by implementing a tool based on a wrapper between the software and the rendering engine that tracked the ids of function calls.

7.2.2 Sampling selection bias

Intuitively, we may expect that the amount of highly detailed features present in the real-world domain would yield a higher difficulty for the instance segmentation task in that domain compared to the difficulty of that task in the synthetic domain. And yet, the performances between the synthetic models on the real-world test set and the real-world control models synthetic test set gives a conflicting suggesting. The synthetic models performs better at mAP@[50:70] and much better at mAP@[75:90] in the real-world test set compared to the corresponding performance of real-world control models on the synthetic test set. This may be explained by the sampling selection bias between the domains (see Section 5.2.3), being that the domain-invariant features are distributed differently between the domains. Based on the visual comparisons of the training -and test sets done in Section 5.2.3 we noted that there was a difference in the distributions of domain-invariant features between the synthetic and real-world domain in the respective synthetic and the real-world datasets. It seemed like that the synthetic datasets contained both the same domain-invariant geographic features as the real-world datasets in addition to other domain-invariant geographic features not present in the real-world datasets. In particular, this was the case for geographical features such as range between camera and objects. It seemed like the common domain-invariant features in the synthetic and real-world datasets were those from short to moderate range, while the non-common domain-invariant features in the synthetic dataset were those from moderate to long-range.

In conclusion, based on the performance differences and the sampling selection bias, it seems likely that the real-world test set is less challenging than the synthetic test set. The same goes for the real-world training set, as it is very similar to the test set. This may have given the models trained in the synthetic domain, and possibly the real-world adapted models with synthetic models as source models, an advantage when they were tested on the real-world dataset. This is because they can be expected to have become general compared to the case if the domain-invariant distributions had been equal. If this is the case, then this is an advantage that the control models have not had. However, this advantage should not be viewed as an error in the experimental design. Instead, it should be view as an advantage and a positive implication given by the use of synthetic data, since it allowed for a more general dataset than what was possible to acquire from the real-world.

Consequently, it may be expected that the performance of the synthetic and adapted models would drop if the real-world test sets were more representative of their domain as the dataset would become more diverse. Likewise, it may be expected that the performance of the real-world control models on the synthetic test set would increase if the real-world training-set were to be more representative of its domain and share more of the domain-invariant features with the synthetic test set.

7.2.3 Limitations from only using a synthetic validation set

A critical observation was made in Section 6.2 where it was seen that scores stagnated on the synthetic test set between stage two and three of the gradual fine-tuning approach, while scores continued to improve on the real-world test set. Hence, gradual fine-tuning

was continually capable of learning cross-domain features from the synthetic data despite this not leading to increased performance on the synthetic test set. It is not possible to tell how long this would have lasted had the training proceeded for longer. However, since gradual fine-tuning was stopped due to stagnation in loss improvement based on the synthetic validation-data (see Figure 5.6 in Section 5.2.8), it is a likely possibility. Furthermore, this could also be the case for full fine-tuning as well, but since only one model was tested, it is not possible to tell. However, since full fine-tuning was stopped due to overfitting, it is less likely. Nevertheless, if it were to be the case despite overfitting in the synthetic domain, this would have made the observation even more critical. A theory is that small improvements in the models in the synthetic domain, even almost unobservable, give noticeable improvements in the real-world domain given the poor performance in that domain. Hence, only small improvements may be required for having a noticeable impact on performance.

These results strongly suggest that validations scores should be calculated for both synthetic data and real-world data since this phenomenon is not observable when only using synthetic data. In addition, it suggests that there may be a compromise between overfitting in the synthetic domain and increased cross-domain knowledge when using a two-stage source-target domain adaptation approach. However, cross-domain knowledge is though to be positively co-related to generalization, which stands in opposition to overfitting. Therefore, this suggestion stands as a basis for further research on the topic.

As a last note, the remaining potential for cross-domain knowledge in the source model may have had an effect of the performance impact from using synthetic data, as the gradually fine-tuned all-layers model was used as a source model for the adapted models despite not having reached its maximum cross-domain knowledge potential.

7.2.4 Performance impact

We have seen that all adapted models persistently achieved better performance than the corresponding control models. The difference was most visible between the fully fine-tuned adapted and control models, while the difference between the gradually fine-tuned models decreased as more layers were opened for fine-tuning (see the line-plot in Figure 6.1). Since the training of all models lasted until there was a stagnation in loss improvement or before overfitting became present, it cannot be expected that there could be any further improvement had the training been extended. Hence, it is clear that the use of a source model which was trained on synthetic data have had a positive impact on the performance of real-world adapted models that were further fine-tuned with real-world data of limited size. However, while the difference between the fully fine-tuned models was significant with 10.93 percentage-points, the difference between the best performing adapted model and the best performing control model was only 0.86 percentage-points. In addition, the performance between the later two models on the different types of vessels were nearly the same (see precision-recall curves in Figure 6.5). Therefore, the positive impact when considering the real-world performance is noticeable, but small. Conversely, it is possible that the impact is sub-optimal, considering the discussion in Section 7.2.3

While this is the case, we have seen that all the adapted models maintained their knowledge about the synthetic domain, were they outclassed the control models. This can partly be credited to the sampling selection bias as discussed in the previous section, but nevertheless, it seems that the adapted models contains more knowledge about the maritime domain than the control models. Certainly, this may helps towards generalization and robustness at a level that the control models cannot match without increased diversity in the real-world training-set. Hence, the use of synthetic data have had a positive impact beyond increased performance score on the real-world test set.

7.3 Domain adaptation

We have seen that the gradual fine-tuning approach overall achieved better performance than the full fine-tuning approach, both in terms of transfer learning phase for the synthetic models and for the real-world control models, and in terms of domain adaptation for the adapted models. We first discuss an observation regarding the transfer learning phase and two observation regarding the domain adaptation phase, and then we discuss some considerations regarding the real-world training-set augmentation scheme.

We saw during the transfer learning phase that the full fine-tuning approach was consistently inferior to the gradual fine-tuning approach as the progressive opening of layers proceeded. Complementary to this, the same pattern applied to the test set from the unseen domains. However, the difference in performance between the full fine-tuning approach and the gradual fine-tuning approach was significantly higher in the unseen domains compared to the seen domains. This difference was visible already at the first stage of the gradual fine-tuning approach, and became higher as the approach proceeded. This is staggering, because it suggests that the gradual fine-tuning approach has been more capable of learning domain-invariant features than the full fine-tuning approach. The explanation is likely that the domain-unrelated general layers are useful for extracting features from the maritime domain despite not having seen the maritime domain before. After all, this is the main motivation for using a pre-trained source model. Since the gradual fine-tuning approach at first restricts the ability to change the general layers in the source model, the model will have to adapt the tune-able layers to the features extracted by these general layers which preserves them. On the contrary, the full fine-tuning approach, which allows for tuning all layers from the start, will not preserve any general layers and may thus fit stronger to the synthetic domain-exclusive features.

During the domain adaptation phase, the difference between the two approaches was not significant and both were among the top three models on the real-world testset together with the gradually fine-tuned control model. A theory is that since the source model achieved what has been noted as a good performance on the target domain, it is possible that it have reached a local minima which the adapted models remained within, irrespective of the domain adaption approach. It would be interesting to repeat the experiment multiple times to compare the average performance and to see if the small difference persists, but at the current moment it is not a sufficient difference in performance to conclude whether the gradual fine-tuning approach is a better approach for performing in the real-world domain

given the same well-performing synthetic source model. However, while both approaches was capable of maintaining the knowledge about the synthetic domain, the fully fine-tuned adapted model achieved a noticeable higher score at mAP@[75:90]. This is unexpected, considering the reasoning in the last paragraph based on that the gradual fine-tuning approach had a greater capability to maintain general features. It can be that this is a matter of coincidence. Nevertheless, the difference is to small to give any conclusions.

As a final note, we mention that the same data augmentation scheme for the real-world data was used for all adapted models and for all control models due to time restrictions. Thus, we can not tell if the augmentation has hurt or improved the performance of the models on the real-world test set.

7.4 LIME for instance segmentation

Since instance segmentation involves masking of object, this task is in it self an explanation. While that is the case, the experiment demonstrated that LIME could be used to obtain even finer explanations based on the parts of the mask and the surrounding area. After having examined the resulting explanations, they were considered representative of the models behaviour at the locality of the images used in the experiment. As expected, there was variations between the two types of explanations. Yet, these variations was considered reasonable given the scoring schemes. However, it was noted that the mask scoring scheme had weaknesses because it would give unjustifiably low scores even for optimal mask predictions given specific cases of image perturbations. Besides, the scoring of masks what not based on any mask quality score given directly by the model. Therefore, the mask prediction explanation was not a direct explanation of the model's predicted mask, but rather an explanation of what part of the vessel which was important for obtaining similar masks as the original predicted mask. If the model had given a confidence for the mask prediction in the same manner as the detection or classification confidence, the method would have the ability to explain the mask prediction directly. Likewise, the detection confidence explanation was affected by the relationship between the predicted mask and the original mask, since all detections with a mask overlap of less than 2.5 % was omitted. However, this was justified since non-overlapping detections could be assumed to be non-relevant for the detection.

In was noted that explanations showed that the model valued unique characteristic of each vessel type, but did not show noticeable relationship between those types of characteristics and general vessel characteristics. This did not assert any trust in that the model would be able to discriminate between vessels and other object with the same characteristics, e.g. harbours and ports. The explanation approach can be extended to include multiple sets of segmentation per image, where some sets separates the unique and general vessel characteristics, while other sets combines some of these characteristics. If explanations based on segments with some combined characteristics values these combinations to a larger extend than the explanation based on segments with only separated characteristics, there could be a ground for saying that the model also considers general vessel characteristics important. This would assert more trust than what was obtained in the experiment.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

In this thesis we have presented a method developed for acquiring instance-level labeled synthetic images from Kongsberg Digital's maritime simulator named K-Sim, explored the implication of using synthetic data and how its usage and domain adaptation by full and gradual fine-tuning affects the performance of Mask R-CNN, and explored how a recent method within Explainable AI named LIME can be used for obtaining explanations from instance segmentation models. We conclude the findings in a chronological and point-wise manner:

- The method developed for acquisition of synthetic data from a simulator proposes a way to label objects in images obtained by computer graphics without access to the inner workings of the software given the user interface with a visible clock, access to the model library, and separability between colors of manipulated texture of object that are to be labeled and colors of other elements such as the background. Furthermore, the method shows how a combination of thresholding, morphological transformation, calculation of connected components, probabilistic clustering using Variational Bayesian Gaussian Mixture Models (VBGMM) and subsequent agglomerative hierarchical clustering could be used to cluster regions of strong colors in an image. Therefore, the method is a source to inspiration for image labelling techniques. Besides, the probabilistic hierarchical clustering method in it self is also a source to knowledge in the field of machine learning.
- The use of a synthetic source model for the domain adaptation showed a consistently positive impact on the performance in the real-world. In addition, the experiment suggested that pure synthetic models trained on heavy augmented synthetic images were able to learn a greater portion of domain-invariant features than the corresponding pure real-world models trained on augmented real-world images. This is

explained by positive implication from using synthetic data; simplistic representation of the synthetic domain highlighted domain-invariant features, use of synthetic data provided far greater numbers of images and vessels, and that simulation setup freedom gives greater diversity than what is possible to obtain in the real-world dataset. In addition, a suitable augmentation scheme is believed to have eased domain adaptation by increasing diversity, highlighting features, and closing the gap between the synthetic and real-world dataset.

- Inspection and comparison between the precision-recall curves from the best performing adapted model and control model suggested that the adapted models had brought a class-based bias from the synthetic dataset due to class imbalance, despite there being a class balance in the real-world dataset. This observation shows that dataset imbalance can have detrimental effects across domain adaptation.
- The results suggest that when the source and target domain are very different (MS COCO dataset and K-Sim synthetic dataset), and when the source and target tasks are completely different which requires replacement of the overheads, gradual fine-tuning beyond the overheads is superior to full fine-tuning. Given the same conditions, the results strongly suggests that gradual fine-tuning, even for overheads only, is far more capable of learning domain-invariant features in the target domain than full fine-tuning.
- The results suggests that when the source and target domain shares a high but unknown portion of domain-invariant features, and when a source model has achieved a good performance in the target domain (mAP@[50:70] $\approx>$ 75 % and mAP@[75:95] $\approx>$ 43 %), there is little difference between gradual and full fine-tuning.
- The results indicates that when the source and target domain shares a high but unknown portion of domain-invariant features, fine-tuning during gradual fine-tuning in the source domain can continue to improve performance in the target domain after it has stagnated in the source domain. It can neither be concluded nor debunked that this is possible for full fine-tuning as well.
- LIME has been shown to able to give explanations for detection by instance segmentation models beyond the explainable value of the mask prediction.

8.2 Future Work

Much work has been put into implementing the experimental framework, and therefore, little effort is required to expand on and beyond the objectives of this thesis. The following is an overview over proposals for further work to utilize the experimental framework, as well as other research ideas.

- The image acquisition method in Chapter 4 can be improved in multiple ways. Some proposals has been been in the chapter, and has been concerned about embedding the thresholding into the VBGMM clustering. Further improvements can be based on utilizing both images in pairs of synchronized images by obtaining region proposals from the difference image between the two.

- The experiment should be replicated without augmentation of the real-world data. Since the aim has been to experiment has been to study the implications of using synthetic data for training of instance segmentation models, augmentation of real-world data adds an additional and unnecessary element which may distort the ability to interpret results directly related to the use of synthetic data.
- The experiment should expand on the topic of synthetic data augmentation. In addition, more advanced DA techniques[31] could be explored and implemented.
- The thesis has just briefly touched on the topic of Explainable AI. It has been proposed to improve the dedication of LIME to instance segmentation by improving the scoring of predicted masks based on original predicted masks. Furthermore, the implementation of Mask R-CNN used in this thesis provides the ability to obtain saliency map from the different layers of the architecture and access to an complete overview over region proposals, which may provide insight into its workings. In addition, since the implementation is based on Tensorflow, and there exist multiple libraries for running different backwards pass algorithms in order to obtain different saliency maps which integrates this software library, it would be very interesting to see if it is possible to pass predicted masks backwards into the architecture, and to see what this may reveal.
- K-Sim is used for education of professional personnel in maritime operations. A research proposal is to see if and how the virtual world of the simulator can be made available for AI actors.

Bibliography

[1] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[2] A. Dutta, A. Gupta, and A. Zissermann. VGG image annotator (VIA). http://www.robots.ox.ac.uk/ vgg/software/via/, 2016. [Online; version: 3.0.4, accessed 02-Mai-2019].

[3] Abhishek Dutta and Andrew Zisserman. The VGG image annotator (VIA). *CoRR*, abs/1904.10699, 2019.

[4] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *CoRR*, abs/1611.03530, 2016.

[5] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *CoRR*, abs/1411.1792, 2014.

[6] Pierluigi Salvo Rossi. Synthetic data and computer vision. `https://brainntnu.no/2019/03/01/synthetic-data-cv/`, March 2019. [Online; accessed at 01-June-2019].

[7] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, March 1986. DOI:10.1007/BF00116251.

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[9] Stuart J. Russell and Peter Norvig. *Artificial intelligence - a modern approach, 2nd Edition*. Prentice Hall, 2003.

[10] Gaussian mixture model. brilliant.org. `https://brilliant.org/wiki/gaussian-mixture-model/`. [Online; accessed 09-June-2019].

[11] David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. Variational Inference: A Review for Statisticians. *CoRR*, abs/1601.00670, 2018.

[12] Waleed Abdulla. Mask r-cnn for object detection and instance segmentation on keras and tensorflow. `https://github.com/matterport/Mask_RCNN`, 2017. [Online; accessed March-2019].

[13] Li Liu, Wanli Ouyang, Xiaogang Wang, Paul W. Fieguth, Jie Chen, Xinwang Liu, and Matti Pietikäinen. Deep learning for generic object detection: A survey. *CoRR*, abs/1809.02165, 2018.

[14] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

[15] Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015.

[16] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[17] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017.

[18] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[19] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015.

[20] Cs231n: Convolutional neural networks for visual recognition 2018. `http://cs231n.stanford.edu/`, 2019. [Online; accessed 25-February-2019].

[21] Michael a. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015. [Online; accessed February 26, 2019].

[22] Approximation with artificial neural networks. *Faculty of Sciences, Etvs Lornd University, Hungary*, 2001. MSc. thesis (2001) doi:10.1.1.101.2647.

[23] Guido Montufar, Razvan Pascanu, Kyunghyun Cho, and Y Bengio. On the number of linear regions of deep neural networks. *NIPS 2014*, pages 2924–2932, 02 2014.

[24] Jurgen Mayer, Khaled Khairy, and Jonathon Howard. Drawing an elefant with four complex parameters. *American Journal of Physics, v.78, 648-649 (2010)*, 78, 05 2010.

[25] Kaiming He and Jian Sun. Convolutional neural networks at constrained time cost. *CoRR*, abs/1412.1710, 2014.

[26] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[27] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105. 2012.

[28] Unknown. Image aquried from webpage with url:. `https://www.slideshare.net/windmdk/mask-rcnn`. [Online; accessed 22-June-2019].

[29] Tsung-Yi Lin, Piotr Dollár, Ross B. Girshick, Kaiming He, Bharath Hariharan, and Serge J. Belongie. Feature pyramid networks for object detection. *CoRR*, abs/1612.03144, 2016.

[30] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, Oct 2010.

[31] Magnus Reiersen. Deep visual domain adaptation: From synthetic data to the real world. *Master thesis, NTNU*, 2018.

[32] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *CoRR*, abs/1411.1792, 2014.

[33] Brian Chu, Vashisht Madhavan, Oscar Beijbom, Judy Hoffman, and Trevor Darrell. Best practices for fine-tuning visual classifiers to new domains. In *ECCV Workshops*, 2016.

[34] Stephan R. Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. *CoRR*, abs/1608.02192, 2016.

[35] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. *CoRR*, abs/1604.01685, 2016.

[36] Aleksandar Zlateski, Ronnachai Jaroensri, Prafull Sharma, and Fredo Durand. On the importance of label quality for semantic segmentation. pages 1479–1487, 06 2018. Doi:10.1109/CVPR.2018.00160.

[37] Finale Doshi-Velez, Mason Kortz, Ryan Budish, Chris Bavitz, Sam Gershman, David O'Brien, Stuart Schieber, James Waldo, David Weinberger, and Alexandra Wood. Accountability of AI under the law: The role of explanation. *CoRR*, abs/1711.01134, 2017.

[38] David Gunning. Explainable arificial intelligence (xai). `https://www.darpa.mil/attachments/XAIProgramUpdate.pdf`, 2017.

[39] Matt Turek. Explainable arificial intelligence (xai). `https://www.darpa.mil/program/explainable-artificial-intelligence`.

[40] Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. Methods for interpreting and understanding deep neural networks. *CoRR*, abs/1706.07979, 2017.

[41] Bryce Goodman and Seth Flaxman. European union regulations on algorithmic decision-making and a "right to explanation". *CoRR*, abs/1606.08813, 2017.

[42] Lisa Anne Hendricks, Zeynep Akata, Marcus Rohrbach, Jeff Donahue, Bernt Schiele, and Trevor Darrell. Generating visual explanations. *CoRR*, abs/1603.08507, 2016.

[43] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. *CoRR*, abs/1602.04938, 2016.

[44] Thomas Lin Pedersen and Michaël Benesty. Understanding lime. `https://cran.r-project.org/web/packages/lime/vignettes/Understanding_lime.html`. [Online; accessed 14-June-2019].

[45] Waleed Abdulla. Lime: Explaining the predictions of any machine learning classifier. `https://www.oreilly.com/learning/introduction-to-local-interpretable-model-agnostic-explanations-lime`. [Online; accessed 14-June-2019].

[46] Shipspotting.com, maritime image repository. `http://www.shipspotting.com`. [Online; accessed between 15-April-2019 and 16-Mai-2019].

[47] Vesseltracker/vessels.com, maritime image repository. `https://www.vesseltracker.com/en/vessels.html`. [Online; accessed between 02-Mai-2019 and 16-Mai-2019].

[48] Tante Manfred. Ship spotting. `https://www.youtube.com/channel/UCXsqqvEuQzYrRJIrFW4EG0g`. [Online; accessed between 02-Mai-2019 and 16-Mai-2019].

[49] Fred Vloo. Ship spotting. `https://www.youtube.com/user/fredvloo`. [Online; accessed between 02-Mai-2019 and 16-Mai-2019].

[50] cnlhwk. Ship spotting. `https://www.youtube.com/user/cnlhwk/videos`. [Online; accessed between 02-Mai-2019 and 16-Mai-2019].

[51] Alexander B. Jung. imgaug. `https://github.com/aleju/imgaug`, 2018. [Online; accessed spring-2018].

[52] ngrok. `https://ngrok.com`. [Online; accessed between 02-Mai-2019 and 17-June-2019].

[53] Quick guide to run tensorboard in google colab. `https://www.dlology.com/blog/quick-guide-to-run-tensorboard-in-google-colab/`. [Online; accessed between 02-Mai-2019 and 17-June-2019].

[54] Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, and Tony Yu. scikit-image: Image processing in python. *CoRR*, abs/1407.6245, 2014.

[55] Mateusz Buda, Atsuto Maki, and Maciej A. Mazurowski. A systematic study of the class imbalance problem in convolutional neural networks. *CoRR*, abs/1710.05381, 2017.

Appendix A

Listings

```
import imgaug as ia
from imgaug import augmenters as iaa

seq = iaa.Sequential([
    iaa.Fliplr(0.5),

    iaa.Sometimes(0.7,
        iaa.SomeOf((1,4),[
            iaa.GammaContrast(gamma = (0.5,1)),
            iaa.Add((-30,30)),
            iaa.Add((-25,25), per_channel = 0.5),
            iaa.AddToHueAndSaturation((-20,20))
        ]),
        iaa.Sometimes(0.16666,
            iaa.Grayscale(alpha = (0.4,0.8))),
    ),

    iaa.Sometimes(0.5,iaa.SomeOf((1,2),[
        iaa.SimplexNoiseAlpha(iaa.EdgeDetect(np.random.randint(1,3)/5)),
        iaa.JpegCompression(compression = [0,70]),
        iaa.GaussianBlur(sigma = (0.25,3.0)),
        iaa.AdditiveGaussianNoise(scale = (0.01*255,0.06*255)),
        iaa.Emboss(alpha=1.0,strength = (0,0.5)),
        iaa.Sharpen(alpha=(0,1.0),lightness = (0.75,1.5)),
    ])),

    iaa.Sometimes(0.50,iaa.OneOf([
        iaa.CoarseDropout(p = (0.005,0.05), size_percent = (0.1,0.05)),
        iaa.CoarseSalt(p = (0.005,0.05), size_percent = (0.1,0.05)),
        iaa.CoarsePepper(p = (0.005,0.05), size_percent = (0.1,0.05)),
        iaa.CoarseSaltAndPepper(p=(0.005,0.05),size_percent = (0.1,0.05)),
```

```
        iaa.Snowflakes(density=(0.005,0.015),flake_size=(0.8,1),speed=0),
    ]))
], random_order = True)
```

Listing A.1: Augmentation scheme for synthetic data

```
import imgaug as ia
from imgaug import augmenters as iaa

augmentation = iaa.Sequential([
    iaa.Fliplr(0.5),

    iaa.OneOf([
        iaa.Affine(
            scale={"x": (0.8,1.0), "y": (0.8,1.0)},
            translate_percent={"x": (-0.1,0.1), "y": (-0.1,0.1)},
            rotate=(-3,3),
            shear=(-3,3),
            order=[0,1],
            cval= 0,
            mode= "constant",
        ),
        iaa.Sequential([
            iaa.SomeOf((1,4),[
                iaa.PerspectiveTransform(scale = (0.0,0.15),
                                         keep_size = True,
                                         cval = 0,
                                         mode = "constant"),
                iaa.Pad(percent = ((0,0.5),(0,0.5),(0,0.5),(0,0.5)),
                        pad_mode = "constant",
                        pad_cval = 0,
                        keep_size = True),
                iaa.Crop(percent = ((0,0.2),(0,0.2),(0,0.2),(0,0.2)),
                         keep_size = True),
            ], random_order = True),
            iaa.Affine(rotate = (-2,2), cval = 0, mode = "constant")
        ]),
        iaa.Sequential([
            iaa.SomeOf((1,4),[
                iaa.Pad(percent = ((0,0.5),(0,0.5),(0,0.5),(0,0.5)),
                        pad_mode = "constant",
                        pad_cval = 0,
                        keep_size = True),
                iaa.Crop(percent = ((0,0.3),(0,0.3),(0,0.3),(0,0.3)),
                         keep_size = True)
            ], random_order = True),
            iaa.Affine(rotate = (-2,2), cval = 0, mode = "constant")
        ]),
    ]),

    iaa.SomeOf((1,2),[
        iaa.GammaContrast(gamma = (0.5,1.2)),
        iaa.Add((-15,20), per_channel=0.5),
        iaa.AddToHueAndSaturation((-20,20))
    ]),
], random_order = True)
```

Listing A.2: Augmentation scheme for real-world data

```
!ln -sf /opt/bin/nvidia-smi /usr/bin/nvidia-smi
!pip install gputil
!pip install psutil
!pip install humanize
import psutil
import humanize
import os
import GPUtil as GPU

GPUs = GPU.getGPUs()
gpu = GPUs[0]
def printm():
 process = psutil.Process(os.getpid())
 print("Gen RAM Free: " + humanize.naturalsize( psutil.virtual_memory().
     available ), " | Proc size: " + humanize.naturalsize( process.
     memory_info().rss))
 print("GPU RAM Free: {0:.0f}MB | Used: {1:.0f}MB | Util {2:3.0f}% | Total
      {3:.0f}MB".format(gpu.memoryFree, gpu.memoryUsed, gpu.memoryUtil*100,
      gpu.memoryTotal))
printm()
```

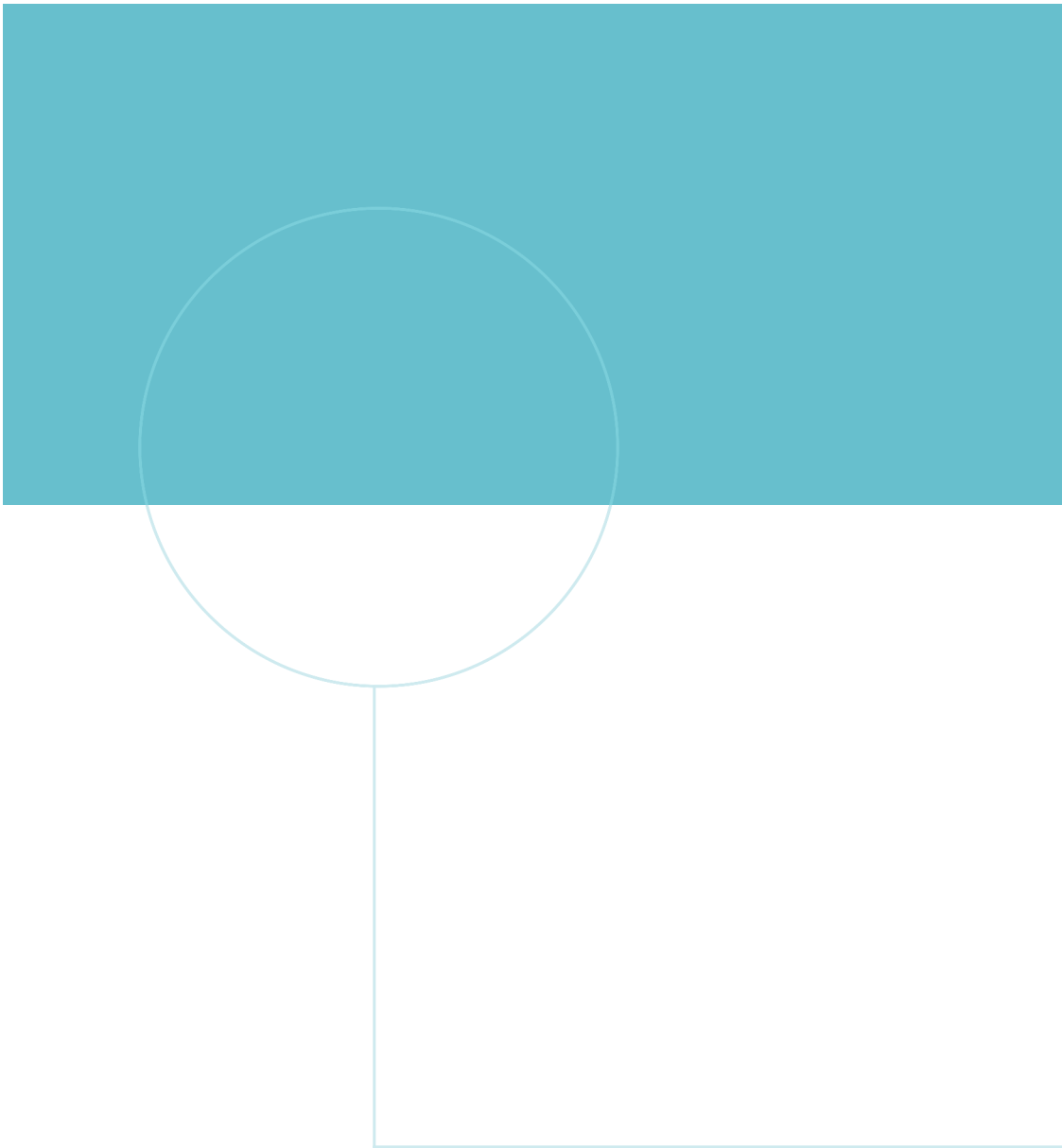
Listing A.3: Script for monitoring GPU load in Google Colaboratory

```
LOG_DIR = '/content/KSim_Mask_RCNN/train_own_dataset/logs/k-
    sim20190604T1408'

! wget https://bin.equinox.io/c/4VmDzA7iaHb/ngrok-stable-linux-amd64.zip
! unzip ngrok-stable-linux-amd64.zip

get_ipython().system_raw('tensorboard --logdir {} --host 0.0.0.0 --port
    6007 &'.format(LOG_DIR))
get_ipython().system_raw('./ngrok http 6007 &')
! curl -s http://localhost:4041/api/tunnels | python3 -c \
    "import sys, json; print(json.load(sys.stdin)['tunnels'][0]['
    public_url'])"
```

Listing A.4: Script for tunneling of Tensorboard Dashboard from Google Colaboratory by the use of ngrok [53]



NTNU
Norwegian University of
Science and Technology