

Simen Viken Grini

# Object Detection in Maritime Environments

Systematic Training and Testing of  
Deep Learning-based Detection Methods for  
Vessels in Camera Images

Master's thesis in Engineering Cybernetics

Supervisor: Edmund Brekke

January 2019



Simen Viken Grini



Simen Viken Grini

# Object Detection in Maritime Environments

Systematic Training and Testing of  
Deep Learning-based Detection Methods for  
Vessels in Camera Images

Master's thesis in Engineering Cybernetics  
Supervisor: Edmund Brekke  
January 2019

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



Norwegian University of  
Science and Technology



---

# Preface

The work presented in this report is the result of the term project for the course "TTK4900 - Engineering Cybernetics, Master's Thesis", conducted in the last semester of the Master of Science program at the Norwegian University of Science and Technology (NTNU), Department of Engineering Cybernetics.

I would like to thank my supervisor Edmund Brekke and my co-supervisor Håkon Hagen Helgesen for their valuable feedback, guidance and support throughout the work with this project. Edmund Brekke has also contributed with important data, for which I am very grateful.

The discussions and support from other students with whom I shared office have been very helpful. While going through thousands of images of slightly different boats with slightly different rectangles around them, the work environment, chess games and jokes have been crucial to keep my sanity on an acceptable level.

Last, but not least, I would like to thank my family for their support during my studies in Trondheim. A special thanks to my father, Frode, who contributed to this thesis with helpful constructive criticism. A special thanks also goes to my grand parents, Jorunn and Ronald, for their hospitality during the work with this thesis. Finally, a big thanks to my brother, Jonas, for his magnificent MS Paint skills, and my uncle, Rolf-Einar, for his proofreading.

Oslo, 11 January 2019

Simen Viken Grini

---

---

---

---

# Abstract

Machine learning methods based on deep convolutional neural networks (CNN's) have in recent years come to dominate the state of the art in object recognition. Huge research efforts are currently being devoted to this field, and new network architectures are frequently being proposed. For the particular task of detection of ships in images, these networks must be trained on an appropriate set of annotated images. As these tasks, which are somewhat labor intensive, will have to be repeated many times as new networks and improvements appear, a systematic and reusable framework is highly desirable. To develop this is the goal of this master thesis. In particular, the thesis focuses on the following:

- Conduct a literature survey on methods for detection of ships in camera images. In particular, the survey should discuss the rationale for the recent popularity of deep learning techniques, include the most popular CNN approaches, and discuss other machine learning approaches.
- Develop a complete framework for training and testing, so that both training and testing on new image sets can be done with minimal efforts for popular networks. While the training can accept unordered collections of individual images, it is important that the testing can work on video streams, so that temporal detection metrics can be investigated.
- Train and test the detection algorithms Yolov3 and SSD Mobilenet on real data
- Write report and discuss strengths and weaknesses for both of the networks, and also for different training and testing sets.

The results from the tests done in this project are promising, and shows that boat detection with camera can be reliable in certain situations. Though the results vary for different test data, tests show that detection algorithms like SSD and Yolo can be able to detect close to 100 percent of the boats in a test set when trained properly.





---

# Sammendrag

Maskinlæringsmetoder basert på konvolusjonelle nevralt nettverk (CNNs) har i senere år tatt over som den dominerende teknikken innenfor objekt-deteksjon. Store forskningsressurser blir brukt på å utvikle dette feltet videre, og det kommer stadig nye nettverksarkitekturer og metoder. For å kunne detektere båter i bilder må disse nettverkene trenes på et passende sett med merkede bilder. For å kunne sammenligne deteksjonsmetoder og forskjellig treningsdata bør disse nettverkene kunne trenes enkelt gjennom et gjenbrukbart og systematisk rammeverk. Å utvikle et slikt rammeverk er målet med denne masteroppgaven. Mer spesifikt ser denne oppgaven nærmere på følgende:

- Gjennomføre litteraturstudie på eksisterende metoder for deteksjon av båter i bilder, herunder å diskutere de mest populære CNN-metodene og andre maskinlæringsmetoder.
- Utvikle et rammeverk for trening og testing, slik at denne treningen og testingen kan bli gjort for de mest populære deteksjonsmetodene med minimal innsats.
- Trene og teste deteksjonsalgoritmene YoloV3 og SSD Mobilenet på relevante data.
- Skrive rapport og diskutere styrker og svakheter for både Yolo og SSD, på forskjellige trenings- og testsett.

Resultatene fra dette arbeidet er lovende, og viser at båtdeteksjon med kamera kan være pålitelig i gitte situasjoner. Selv om resultatene varierer for forskjellig testdata, viser testene at deteksjonsalgoritmer som Yolo og SSD kan detektere nærmere 100 prosent av båtene i et testsett når algoritmene er trent riktig.

---

# Table of Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Neural Networks</b>	<b>3</b>
2.1	Artificial Neural Networks . . . . .	3
2.1.1	Training of Neural Networks . . . . .	4
2.2	Convolutional Neural Networks . . . . .	4
2.2.1	Convolutional Layers . . . . .	5
2.2.2	Pooling Layers . . . . .	6
2.3	Transfer Learning . . . . .	6
<b>3</b>	<b>Object Detection</b>	<b>9</b>
3.1	Sliding Window (OverFeat) . . . . .	9
3.2	R-CNN – Regions with Convolutional Neural Networks . . . . .	10
3.2.1	Selective Search . . . . .	10
3.3	Fast R-CNN . . . . .	11
3.4	Faster R-CNN . . . . .	11
3.5	Yolo - You Only Look Once . . . . .	12
3.6	SSD - Single shot Multibox Detector . . . . .	14
3.7	Object Detection in Maritime Environments . . . . .	15
3.7.1	Traditional Computer Vision Methods . . . . .	15
3.7.2	Deep Learning-based Methods . . . . .	15
3.7.3	Challenges in Detection of Maritime Vessels . . . . .	16
3.7.4	Vehicle Detection for Autonomous Cars . . . . .	16
3.7.5	Common Challenges in Maritime Object Detection . . . . .	17
<b>4</b>	<b>Implementation and Datasets</b>	<b>19</b>
4.1	Datasets . . . . .	19
4.1.1	Datasets used for pre-trained weights . . . . .	19
4.1.2	Custom datasets . . . . .	21
4.2	Detection framework . . . . .	24
4.2.1	Hardware . . . . .	25
4.2.2	Google Cloud . . . . .	25
4.2.3	Cloud Instance Setup . . . . .	25

---

4.2.4	Training Yolo . . . . .	26
4.2.5	Training SSD . . . . .	28
4.2.6	Testing with Cloud Detection Framework . . . . .	29
<b>5</b>	<b>Evaluation and Testing Methodology</b>	<b>31</b>
5.1	Precision and Recall . . . . .	31
5.2	Bounding Box Evaluation . . . . .	32
5.3	Precision/recall Curves . . . . .	32
5.4	Average precision . . . . .	33
5.5	Confusion Matrix . . . . .	34
<b>6</b>	<b>Results</b>	<b>35</b>
6.1	Training Sets and Detection Models . . . . .	36
6.2	Results Overview . . . . .	36
6.2.1	Average Precision . . . . .	36
6.2.2	Confusion Matrices . . . . .	37
6.2.3	Main results . . . . .	38
6.3	Case Study 1: Effect from training on buildings . . . . .	40
6.3.1	Tested on MooredBoats . . . . .	42
6.3.2	Tested on the BoatsClose and BoatsFar . . . . .	43
6.3.3	Tested on Trondheimsfjorden . . . . .	46
6.4	Case Study 2: Effect of Training on Moored Boats While Testing for Sailing Boats . . . . .	50
6.5	Case Study 3: Video and Temporal Data . . . . .	52
6.5.1	Video from Trondheimsfjorden . . . . .	52
6.5.2	Video from (Kamsvåg, 2018) . . . . .	56
<b>7</b>	<b>Discussion</b>	<b>59</b>
7.1	Dataset diversity . . . . .	59
7.1.1	Non-detected buildings in Trondheimsfjorden . . . . .	62
7.2	Labeling . . . . .	63
7.2.1	Labeling of sail boats . . . . .	63
7.2.2	Using several boat classes . . . . .	63
7.2.3	Detecting boat parts . . . . .	63
7.3	Clustered boats . . . . .	64
<b>8</b>	<b>Conclusion and Future Work</b>	<b>65</b>
8.1	Future Work . . . . .	66
8.1.1	Test on Video and Real-Time Performance . . . . .	66
8.1.2	More Data . . . . .	66
8.1.3	Tracking . . . . .	66
	<b>Appendix A Setting up Google cloud</b>	<b>67</b>
A.1	Creating an account . . . . .	67
A.2	Setting up an instance . . . . .	67
A.3	Setting up and using gcloud command line tool on local computer . . . . .	70
A.4	Accessing an instance from local computer using Gcloud command line tool	70
A.4.1	SSH in to instance . . . . .	70
A.4.2	SCP file to instance . . . . .	71

---

---

A.4.3	SCP repository to instance . . . . .	71
<b>Appendix B Using the Cloud Detection Framework</b>		<b>73</b>
B.1	Training . . . . .	73
B.1.1	Adding datasets and directory structure setup . . . . .	74
B.1.2	Training YOLO . . . . .	74
B.1.3	Training SSD . . . . .	74
<b>Appendix C Example images</b>		<b>75</b>
C.1	SSD <sub>BSM</sub> and SSD <sub>BSMH</sub> on BoatsClose and BoatsFar . . . . .	76
C.2	Yolo <sub>BSM</sub> and Yolo <sub>BSMH</sub> on BoatsClose and BoatsFar . . . . .	78
C.2.1	Yolo <sub>BSM</sub> specific misclassifications on BoatsClose and BoatsFar . . . . .	78
C.2.2	Yolo <sub>BSMH</sub> better than Yolo <sub>BSM</sub> on BoatsClose and BoatsFar . . . . .	79
C.2.3	Yolo <sub>BSM</sub> better than Yolo <sub>BSMH</sub> on BoatsClose and BoatsFar . . . . .	80
C.2.4	Yolo <sub>BSM</sub> and Yolo <sub>BSMH</sub> same misclassifications on BoatsClose and BoatsFar . . . . .	81
C.3	SSD <sub>BSM</sub> and SSD <sub>BSMH</sub> on Trondheimsfjorden . . . . .	82
C.3.1	SSD <sub>BSM</sub> specific misclassifications on Trondheimsfjorden . . . . .	82
C.3.2	SSD <sub>BSMH</sub> better than SSD <sub>BSM</sub> on Trondheimsfjorden . . . . .	83
C.3.3	SSD <sub>BSM</sub> better than SSD <sub>BSMH</sub> on Trondheimsfjorden . . . . .	84
C.4	Yolo <sub>BS</sub> and Yolo <sub>BSM</sub> differences . . . . .	85
C.4.1	Yolo <sub>BS</sub> overestimating ship size . . . . .	85
C.4.2	Yolo <sub>BS</sub> detecting same boat multiple times . . . . .	86
<b>Bibliography</b>		<b>86</b>



# Introduction

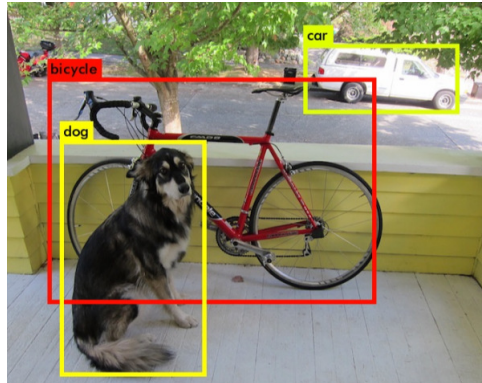
In recent years there has been made substantial progress in the field of autonomous vehicles, and companies such as Google, Tesla and Uber are investing heavily in the development of autonomous cars. In Norway, with our maritime expertise and long coastline, a research field of interest has been autonomous ferries. Currently, Kongsberg Gruppen is developing the world's first zero emission, autonomous container feeder for Yara, which could be able to reduce diesel-powered truck journeys by around 40,000 trips per year (Kongsberg, 2017). In Trondheim, the development of an autonomous passenger ferry is ongoing (Stensvold, 2016).

A crucial part of an autonomous system is the ability to sense and understand its surroundings. Sensors such as lidars and radars are good at measuring distance, and are used both in autonomous cars and ferries. This report investigates how cameras can be used to help an autonomous ferry generate information about its surroundings. The same problem has already been studied to some extent at the Rolls Royce Singapore Lab (Prasad et al., 2016b) and (Prasad et al., 2016c), but it is still necessary to investigate this topic further. This report separates from their research by focusing on detection algorithms based on deep learning.

In the last years, image classification and object detection have developed rapidly. In 2012 there was a breakthrough in image classification when AlexNet won the ImageNet Large-Scale Visual Recognition Challenge by a large margin, using a Convolutional Neural Network (CNN) for the first time in the competition (Krizhevsky et al., 2012). This was achievable because of easier access to the required computational power and the increased amount of data needed for training of such networks. In the following years, several CNNs for image classification were developed, e.g. VGGnet (2014), GoogLeNet (2015) and Microsoft ResNet (2015). Today the best classification algorithms have surpassed human-level performance on the ImageNet dataset (He et al., 2015). This technology can be used for many interesting purposes, e.g. image colorization (Zhang et al., 2016), image captioning (Karpathy and Fei-Fei, 2016) and to create the worlds best GO player (Silver et al., 2016).

The advances in image classification have also led to new deep learning based object detection algorithms. Where image classification seeks to classify objects in an *image*, object

detection aims to localize different objects in an image, *and classify them*. An example of the output of an object detection algorithm is given in figure 1.1.



**Figure 1.1:** Output of an image detection algorithm, image from (Redmon et al., 2016).

This project aims to find a starting point for figuring out how well detection of maritime vessels can be, and to develop a user-friendly system that can be used to test and find statistics for detection algorithms trained on different data. Such a system simplifies identification of challenges in deep learning-based methods, and the work on trying to overcome these challenges can be simplified by having a consistent framework that automates this process. The scope of this thesis involves setting up a framework for testing and training of SSD and Yolo, to build a dataset, and to start the process of finding out what makes an object detector better at identifying maritime vessels in video.

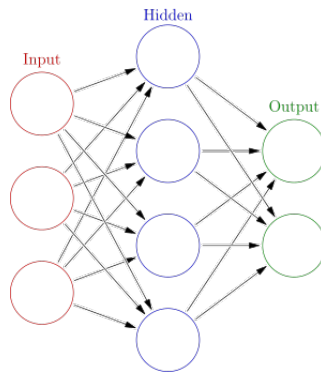


# Neural Networks

Object detection using complex machine learning techniques is possible due to the recent advances in image classification using convolutional neural networks and improvements in hardware. This chapter aims to give the reader a basic understanding of how a convolutional neural network works and how it classifies images, explaining some key concepts, without going into too much detail. A more thorough explanation can be found in (Krizhevsky et al., 2012). The theory presented in this chapter is inspired by (StanfordUniversity, 2018).

## 2.1 Artificial Neural Networks

A neural network is a group of neurons or nodes connected to each other. Each node can receive a signal from its input nodes, process it, and pass it on to its output nodes. In an Artificial Neural Network (ANN), the nodes are structured in layers, where there is one input layer, one output layer, and one or more hidden layers in between. When a node receives a signal, it processes it using an *activation function*, which, given the input, defines the output of the node. The goal is to get the nodes to work together such that the network as a whole gives the correct output. A graphical representation can be seen in figure 2.1.



**Figure 2.1:** Illustration of a simple Neural Network with three layers.

### 2.1.1 Training of Neural Networks

The process of training neural networks is to tune the parameters of the activation function of each individual node such that the final output of the network is correct. A simple example that would conform with figure 2.1, is input nodes where we have different features of a fish, e.g. its weight, color and age. The input nodes pass on the values of the fish features to the hidden layer, which processes the values using the activation function and passes on the value to the output nodes. The output nodes would be the fish type, e.g. salmon and cod. Ideally, if the input is in fact a salmon, we want the value of the salmon output node to get the value 1 and the cod output node to get the value 0. Deviations from this result can be computed as a *loss*. We want to find the parameters for the activation functions in all the nodes that minimizes this loss. This is achieved by training the nodes on data where the class of the input object is known. If we have a data set of 1,000 cods and 1,000 salmons and their weight, color, and age we can use these as inputs, pass it through the network, and look at the loss. Then, by using gradient descent, we can change the parameters in all the activation functions in a way that decreases the loss. This is done using *backpropagation*, which is an algorithm that calculates how the parameters in each activation function should be changed in order to reduce the loss. The idea is that doing this many times makes the network able to classify fish correctly, also on new data.

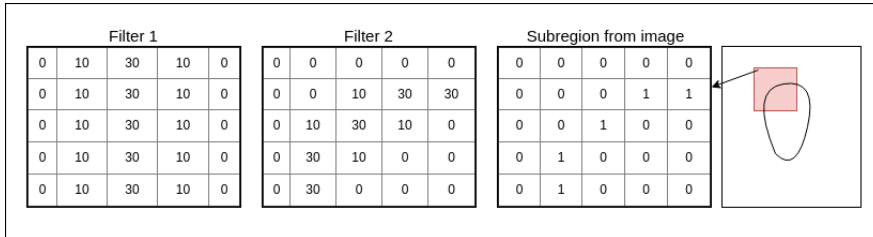
## 2.2 Convolutional Neural Networks

A convolutional neural networks is a type of artificial neural networks, and consist of nodes with activation functions structured into layers. The main difference is that it is assumed that the input is an image, and use this assumption to do operations that are tailored for images, and improve both runtime and accuracy for image classification.

If we want to process an image through an artificial neural network, we would transform the image into a vector, where each row of the image now is in one large row. That would mean that color images, which has three channels (RGB) of size 256 x 256 would have  $256 \times 256 \times 3 = 196,608$  neurons in the first hidden layer. Convolutional neural networks reduces the number of neurons by using *convolutional layers* and *pooling layers*.

### 2.2.1 Convolutional Layers

A convolutional layer consists of a set of filters. When a convolutional layer receives an image input, it slides, or convolves, filters over all the pixels in the input image and output the dot product of the filter and the image at the filters position. This will create an activation map, where we can see the response of the filter at different positions. During training the different filters will change in order to detect different features in the input image. Imagine we want to classify images of handwritten ones and zeros. Some filters may aim to detect the rounded shape of the zeros, and will therefore output a large number if it detects a rounded shape. Figure 2.2 illustrates this.

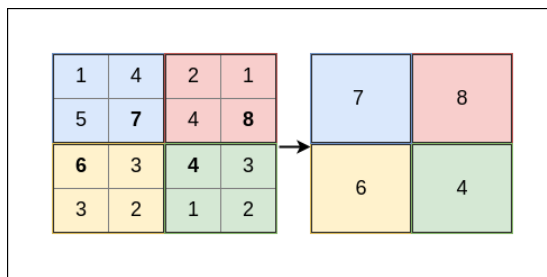


**Figure 2.2:** Filter and subregion of an image of a handwritten zero, in the *Subregion from image* the ones represent pixels that are black, the zeros represent pixels that are white. Filter 1 detects vertical lines, while filter 2 detects upper left regions of circles.

The dot product, or the activation of the two filters at this position in the image is  $(30 \cdot 1) + (10 \cdot 1) + (10 \cdot 1) + (10 \cdot 1) = 60$  for filter 1, and  $(30 \cdot 1) + (30 \cdot 1) + (30 \cdot 1) + (30 \cdot 1) + (30 \cdot 1) = 150$  for filter 2. This means that the activation map for filter 2 is much higher than for filter 1 at this position. For a different subregion the result would be different. When we have several different filters they will activate at different positions and thus have different activation maps. Combining the different activation maps for the different filters tells us something about the image as a whole. During training of a convolutional neural network, the network is trained to draw conclusions of what is in the image based on which filters activate at which positions.

## 2.2.2 Pooling Layers

The pooling layers are downsampling layers. They work by sliding boxes of e.g.  $2 \times 2$  over the pixels of the image, and process the values to output a single value for those 4 pixels. There are different ways to process the pixels, where *max pooling* is a commonly used method. An example of how to downsample an image by a factor of two using max pooling is shown in figure 2.3.



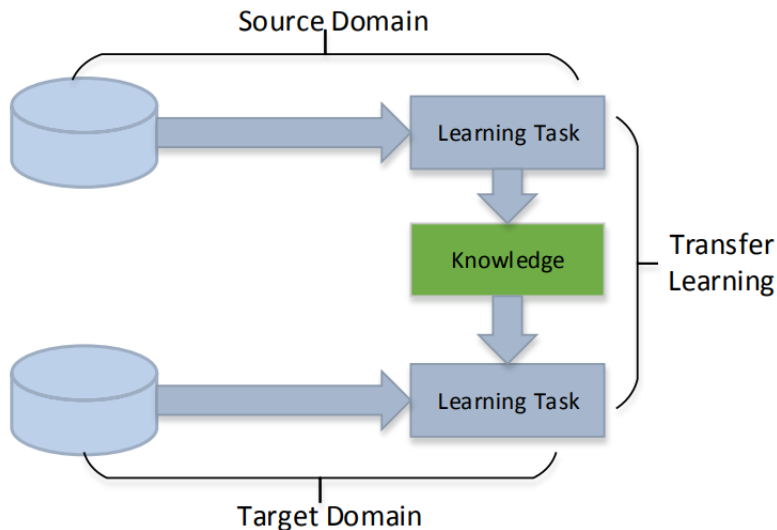
**Figure 2.3:** Max pooling, where a  $4 \times 4$  image is sampled down to a  $2 \times 2$  image. The highest pixel value in each  $2 \times 2$  cell in the left image is the pixel value in the right image.

Max pooling transforms the pixels inside the sliding box into one pixel with the same value as the highest pixel value in the original box. By downsampling the image the number of parameters is reduced, which in turn reduces the computational cost of the algorithm. The pooling layers usually follow a convolutional layer, and can therefore help the next convolutional layer pick up features the previous layer could not. Pooling layers can also help preventing *overfitting*, which is when a classification algorithm performs well on training data, but badly on test data.

## 2.3 Transfer Learning

Transfer learning is a machine learning method where a network trained on one task is repurposed on a second similar task. This is a technique used extensively in this project. A problem in deep learning is *data dependence*, meaning that deep learning processes depends on a massive training dataset to be able to understand the patterns in the data (Tan et al., 2018). This is especially a problem in deep learning compared to traditional machine learning methods, since deep learning methods have to learn features from the dataset, while in traditional machine learning methods the user has to design the features.

To counter the data dependence of a neural network one needs a large well-annotated dataset, which may be both costly and time consuming to acquire (Pan and Yang, 2009). Transfer learning provides a way of utilizing training data that does not conform with the test data. This means that e.g. a classifier trained to detect dogs can be retrained to detect ducks, saving training time and training data requirements compared to training a duck classifier from scratch.



**Figure 2.4:** Transfer learning, figure from (Tan et al., 2018). The goal is to transfer knowledge from a source domain, to a target domain.

Figure 2.4 shows how data from the source domain is used to train a classifier, then using this knowledge in combination with data from the target domain to train a new classifier.

(Tan et al., 2018) categorized network-based deep transfer learning into the following four methods.

#### **Instances-based**

Supplement the training data in the target domain with relevant instances from the source domain by assigning fitting weights to the selected values.

#### **Mapping-based**

Maps instances from the source domain and the target domain to a new data space. The reasoning for this is: "Although there are differences between the two origin domains, they can be more similar in an elaborate new data space." (Tan et al., 2018).

### **Network-based**

Reuses parts of the network that are pre-trained on the source domain. The target domain retrains the last part of the network on new data while keeping the weights in the other layers. The idea is that the first layers in neural network find more general features, while the last layers narrow down to the features specific for the class of interest. A classifier trained on images of boats and a classifier trained on cars may detect similar features in the first layers of the network. This is project uses *pre-trained weights* to further train a boat detector.

### **Adversial-based**

Finds transferable features that are suitable in both target domain and source domain.

# Chapter 3

## Object Detection

The problem of detecting objects in an image is harder than classifying an image only containing one object. In object detection, the goal is both to classify objects and locate the exact location of the object in the image. That makes the problem twofold, first we need to find subregions in the image, and then we need to classify them. Since the latter part of the problem has been solved by CNNs, the part that remains is to find these subregions. An example of detected boats in an image is shown in figure 3.1.



**Figure 3.1:** Detected boats by Yolov3 on image from Trondheimsfjorden.

### 3.1 Sliding Window (OverFeat)

The most straightforward solution to find subregions in an image is to use a sliding window, and OverFeat is a method that applies this technique (Sermanet et al., 2013). The sliding window is a window that convolves over the image, like the filters explained in chapter 2.2.1, and looks at small regions of the image one at a time. This region is then passed through a classification network to determine if this region contains an object of

interest. If the classification network returns a high confidence score, OverFeat considers it as an object.

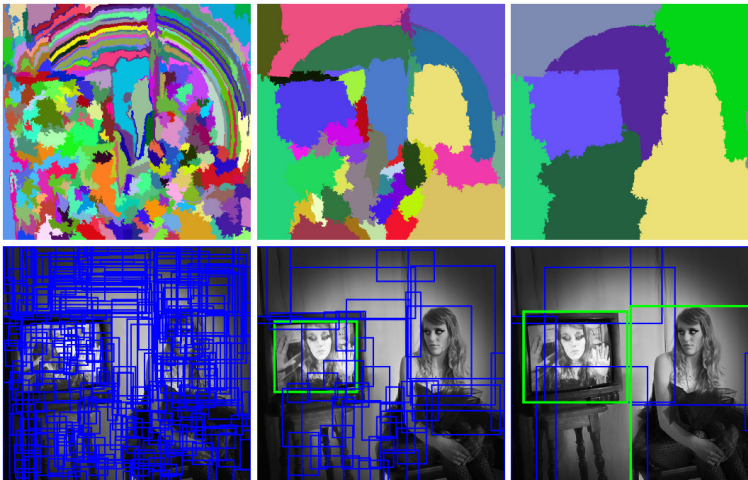
The problem with this approach is that it is very computationally exhaustive, and it is also not scale invariant, meaning we have to try several different scales for the sliding windows to be able to detect objects of different sizes. This means that OverFeat has to classify many subregions, and is therefore not a good option for real-time systems.

## 3.2 R-CNN – Regions with Convolutional Neural Networks

R-CNN (Regions with Convolutional Neural Networks (Girshick et al., 2013)), attacks the problem differently than OverFeat. Instead of trying arbitrary subregions, they propose a solution using the Selective Search algorithm for region proposals.

### 3.2.1 Selective Search

The Selective Search algorithm (Uijlings et al., 2012) groups adjacent pixels based on color, texture, and intensity together and creates segmented pictures as shown in figure 3.2. It is designed to have a very high recall, meaning that it is acceptable to suggest several false positives, as long as it gets all the true positives. False positives are incorrect detections, while true positives are correct detections. Recall is explained further in chapter 5.1.



**Figure 3.2:** Segmented regions shown in top row, and bounding boxes returned shown in bottom row. This shows why different scales of segmentation is important, since bigger objects are only detected when the image is less segmented. Image from (Uijlings et al., 2012).

The Selective Search algorithm takes a segmented image as initial input and performs the following steps:

1. Add bounding boxes corresponding to segmented parts to the list of region proposals. This means that regions with the same color (as shown in figure 3.2) will be



grouped in a bounding box. A bounding box is a rectangular box that surrounds a potential object.

2. Group adjacent segments based on similarity.
3. Go to step 1.

After Selective search has created the region proposals, R-CNN inputs these to a convolutional neural network. The CNN classifies all the region proposals, and if the confidence of the classification is high, the bounding box with its class is outputted from the algorithm.

### 3.3 Fast R-CNN

Two years after the release of R-CNN, Fast R-CNN was released by the same author (Girshick, 2015). In this time he had discovered two drawbacks with R-CNN and found solutions to these issues.

1. R-CNN consists of three different models that all have to be trained. In addition to the CNN, R-CNN consists of a Support Vector Machine (SVM) for classification, and a bounding box regressor, which tightens the bounding boxes around detected objects. Since R-CNN consists of three parts that are trained independently, it makes it harder to train the network as one structure. This is because each submodule is optimizing for themselves and not for the entire system.

#### **Solution**

Instead of training each submodule individually, Fast R-CNN changed the structure in a way that let the whole system be trained together.

2. The CNN has to run once for every region proposal from Selective Search. Since Selective Search is designed to have a very high recall, this ends up being a bottleneck.

#### **Solution**

Instead of calculating the CNN features for every subregion of the image, the CNN features for the entire image are calculated once. Then the features of every region can be extracted from this feature map.

### 3.4 Faster R-CNN

The bottleneck in Fast R-CNN was the region proposals and the Selective Search algorithm. The idea behind Faster R-CNN (Ren et al., 2016) is that the region proposals should depend on the features calculated by the CNN. So instead of using Selective Search, we can use the feature maps outputted from the CNN. The feature maps are then sent into a Region Proposal Network (RPN), which uses a sliding window over the CNN feature map, and outputs  $k$  potential bounding boxes and scores for how correct these boxes are

expected to be. These proposals are then fed into the same classifier as used in Fast R-CNN. The difference between the two is just that the region proposals depend only on CNN features in Faster R-CNN. Thus it bypasses the Selective Search algorithm.

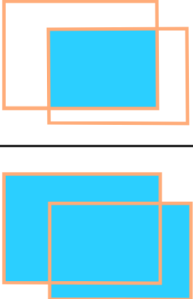
### 3.5 Yolo - You Only Look Once

What separates Yolo (Redmon et al., 2016), (Redmon and Farhadi, 2016) and (Redmon and Farhadi, 2018) from R-CNN methods is that YOLO only looks once at the picture, and there is only one network for the entire detection process. "We reframe object detection as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities. Using our system, you only look once (YOLO) at an image to predict what objects are present and where they are." (Redmon et al., 2016).

Yolo starts by dividing the input image into a  $S \times S$  grid. The goal for each cell is to predict  $B$  bounding boxes and confidence scores for these boxes. If an object has its center in a cell, that cell is responsible for detecting the object. The score of the box says how confident it is that the bounding box contains an object and how accurate it thinks the box is.

$$\text{Confidence score} = P(\text{Object}) \cdot \text{IoU} \quad (3.1)$$

IoU is the *Intersect over Union* between the predicted box and ground truth, shown in figure 3.3. Ground truth is referred to as a manually drawn box over a correct object in the image.

$$\text{IoU} = \frac{\text{Area of overlap}}{\text{Area of union}}$$


**Figure 3.3:** Intersect over Union

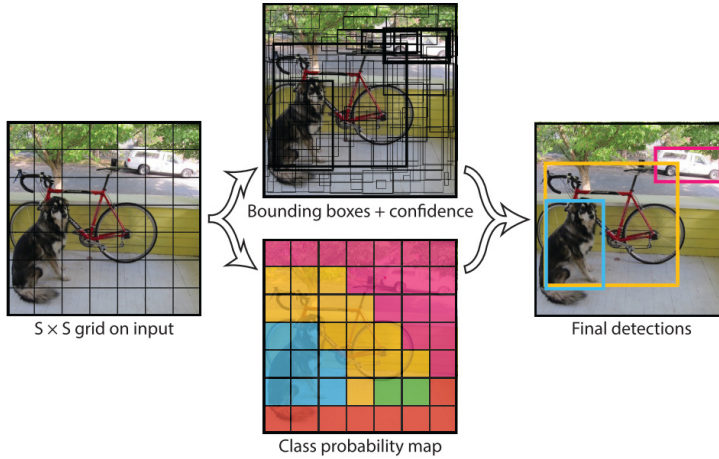
Each grid cell also predicts  $C$  conditional class probabilities, which are conditional on that cell containing an object.

$$P(\text{Class}|\text{Object}) \quad (3.2)$$

For each cell, only one set of class probabilities is calculated. To get class specific scores for each bounding box, we multiply these together.

$$P(\text{Class}|\text{Object}) \cdot P(\text{Object}) \cdot \text{IoU} \quad (3.3)$$

This gives a value of how well the predicted box fits the object and the probability of that class appearing in the box. When we have this value, we can set a threshold, and only use the results that are probable as output. This is shown in figure 3.4.

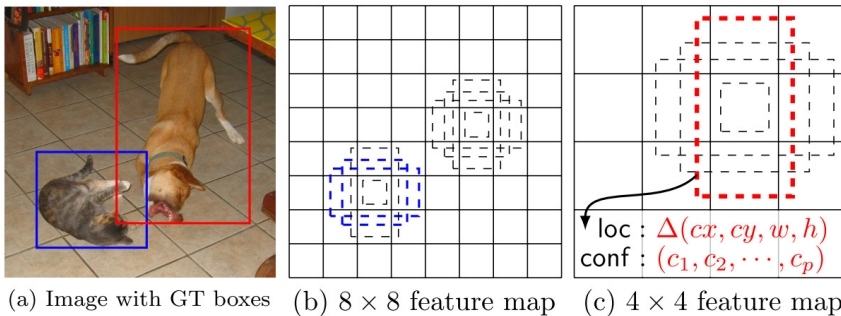


**Figure 3.4:** YOLO flow, Top image shows predicted bounding boxes, the thicker the line the higher the confidence score. The bottom picture shows the class probability map. These combined gives the output image (Equation 3.3), image from (Redmon et al., 2016)

### 3.6 SSD - Single shot Multibox Detector

The Single Shot Multibox Detector works similarly as YOLO compared to Faster R-CNN. SSD does not use a region proposal network, but as YOLO, it discards the proposal generation step and does all the computation in a single network. A difference between YOLO and SSD is that SSD uses different aspect ratios and scales when it grids the input image, which makes it easier to handle objects of different sizes. By circumventing the proposal generation, SSD saves computational time and runs at a frame rate six to eight times higher than Faster R-CNN with similar accuracy, according to (Liu et al., 2016).

Figure 3.5 shows how SSD detects objects with different sizes using different aspect ratios on the feature map. Each cell in the feature maps tries different *anchors*, that is proportional to the cell size. The anchors are the default boxes shown with dotted lines in figure 3.5. The smallest ground truth object in the image was found in the eight by eight grid, while the bigger object is found using the four by four grid, illustrating how different grid sizes detects objects of different size. The anchors in each cell are run through a classifier which gives a confidence score. If multiple boxes predict the same class and have an intersect over union of more than 0.5 the bounding box that has the highest confidence score is chosen. This process is called *non-maximum suppression*.



**Figure 3.5:** SSD training process, image from (Liu et al., 2016).

## 3.7 Object Detection in Maritime Environments

Several approaches have been proposed for solving the challenge of object detection in maritime environments. In this thesis, the methods will be divided into algorithms based on traditional computer vision methods (Traditional CV) and deep learning-based methods.

### 3.7.1 Traditional Computer Vision Methods

#### Background Modelling

Background modeling methods aim to model the background to detect foreground or moving objects. In (Kumar et al., 2015) the authors propose a method for detecting boats using dynamic background modeling and shadow suppression to detect boats. (Rhodes et al., 2007) and (Pires et al., 2010) use background modelling for port surveillance. (Tran and Le, 2016) uses background subtraction to detect moving boats, and saliency detection to detect non-moving boats. These results are finally fused to generate the detected boats in the scene. All these methods rely on a stationary camera to model the background, and is therefore not likely to perform well on an autonomous ferry.

#### Appearance-based Methods

Appearance-based methods seek to model the vessels appearance and to classify it. These methods extract features from the boats using edge-detection, or other traditional CV-based methods, to detect boats. (Eum et al., 2015) uses HOG-like features and background subtraction together to identify and localize boats. (Bloisi et al., 2011) detects boats from a rotatable camera using HAAR-like features. (Dulski et al., 2011) detects small, fast-moving RIBs by using thermal cameras sensitive different bands of the infrared spectre. While these methods have good results in controlled environments, they are susceptible to noise like sun glare, lighting conditions and other non-optimal weather conditions.

### 3.7.2 Deep Learning-based Methods

Earlier in this chapter Faster R-CNN, Yolo and SSD were presented, these are general purpose deep learning-based object detectors. They can be trained on new data, to repurpose them as specialized object detectors, such as a boat detector. There also exist several similar algorithms, and they all can be separated into two categories:

1. Region proposal based methods, which include R-CNN (Girshick et al., 2013), Fast R-CNN (Girshick, 2015), Faster R-CNN (Ren et al., 2016), R-FCN (Dai et al., 2016), Mask R-CNN (He et al., 2018)
2. Regression/classification based methods, which include SSD (Liu et al., 2016), YOLO (Redmon et al., 2016), (Redmon and Farhadi, 2016), (Redmon and Farhadi, 2018), MultiBox (Erhan et al., 2014) and RetinaNet (Lin et al., 2018)

(Zwemer et al., 2018) aims to develop a real-time detection and tracking system for surveillance cameras in harbours. They have gathered a dataset of 48,966 images containing 70,513 ships from several viewpoints and trained a detector based on SSD to perform detection (Liu et al., 2016).

In (Van Etten, 2018), a combination of traditional CV methods and Deep Learning based methods are used to detect boats in satellite images. Large parts of the satellite images do not contain boats, while smaller subframes contains many boats. The solution is to find these subframes by using edge detection methods and then run YOLO. This problem is to some degree relatable to the issue of detecting boats from ships. However, satellite images are less prone to background noise, e.g., from waves. Thus, running the same edge detection algorithms on images taken at sea level returns a lot of noise, and the result is therefore hard to replicate with the same success.

In (Goring, 2017), deep learning-based detection algorithms are trained and tested on detecting maritime navigational markers. In this work they also review how this can be implemented on a boat, with hardware specifications.

### **3.7.3 Challenges in Detection of Maritime Vessels**

In (Prasad et al., 2016a) a thorough review of the challenges in video based object detection in a maritime scenario is presented. The paper focuses on challenges regarding horizon detection, background subtraction, foreground detection and tracking. While this paper gives a good indication of where traditional CV methods has trouble detecting maritime vessels, it does not address the same problem for deep learning-based methods. It discusses how noise in the background, like wakes, foams, clouds, etc. can make the detection process hard. It also discusses how the maritime scene is affected by illumination conditions such as sunlight, twilight conditions, night, rain, haze, fog, etc. and thus making a model suitable for one weather condition useless for a different environment. Similar research done for deep learning-based methods has not been found.

### **3.7.4 Vehicle Detection for Autonomous Cars**

The field of vehicle detection for autonomous cars is in some ways similar to detection of maritime vessels. While the environment around the object of interest is different and has their own unique challenges, there are similarities in how the desired finished product should be.

For vehicle detection a benchmark suite has been developed. (Geiger et al., 2012) provides a platform to "develop novel challenging benchmarks for the tasks of stereo, optical flow, visual odometry / SLAM, and 3D object detection" called KITTI. KITTI simplifies the process of testing new detection algorithms and evaluating performance against previously tested algorithms. It also provides data of which scenarios the detection algorithms perform well, and where it struggles. This gives an indication of how trustworthy the detector will be in a real scenario, also in different environments. On the KITTI website there are links to the highest performing object detectors, with information on how they were implemented.

In (Wedel and Franke, 2007) video is used to verify obstacles, and to find obstacle boundaries, to supplement radar signals for autonomous cars. (Sun et al.) gives a thorough review of the detection methods used for autonomous cars.

### 3.7.5 Common Challenges in Maritime Object Detection

Summed up there are many different approaches to the problem of detecting boats in images. A problem with these articles is that even though they give performance metrics on their dataset, there is no way of concluding that their solution will perform as well in other environments. (Zwemer et al., 2018) for instance, trains an SSD detector for detection of boats, it discusses how the detector performs on boats of different sizes, and it also mentions that it detects fewer boats in "difficult situations such as low light, tiny ships and clutter from in-harbor ships". Still, the only performance statistic provided is how well the detector performs on the entire dataset. This indicates that SSD can be a good option. However, an object detector meant to supplement other sensors on an autonomous ship should have known performance statistics in different environments and under different conditions to be reliable in a real scenario.

For object detectors to be sufficiently trustworthy and implemented on an autonomous ferry there needs to be well known performance statistics. If the performance statistics in for instance foggy conditions are unknown, the object detector is practically useless under these conditions. Even if the detector performs well, there is no way of telling whether the detections are wrong or right if there is no data supporting the robustness of the output.

A platform such as KITTI (Geiger et al., 2012) for detection of maritime vessels could be highly useful to get the required performance statistics. Such a system could be used to investigate how different training data affects the performance of the algorithm in different environments, how different detection algorithms compare to each other, and to set clear benchmarks for future research.





# Chapter 4

## Implementation and Datasets

This chapter covers how the cloud detection framework implemented in this project works, and why it is implemented the way it is. It also covers datasets relevant to this project. A more thorough walkthrough of how to use the cloud detection framework is attached in the appendix. A user manual can also be found in the GitHub repository for this project<sup>1</sup>.

### 4.1 Datasets

The object detection algorithms used in this report all comes with the possibility of transfer learning and training on pre-trained weights. This means that the network has been trained on a dataset and thus already has learned to find features in images and classify them. This is helpful even if the object one wishes to train the algorithm for, is not within the dataset used when optimizing for the pre-trained weights. Using pre-trained weights during training saves computational time for training, and it reduces the required number of training images of the object the detector will be trained on.

#### 4.1.1 Datasets used for pre-trained weights

The detection algorithms used in this project comes with several different pre-trained weights, where the differences include both network configuration and datasets used in training. The datasets presented here are some of the most well-known datasets used in object detection and are also used in the training of the pre-trained weights for both Yolo and SSD, including Imagenet, VOC Pascal and COCO.

##### ImageNet

ImageNet is a "large-scale ontology of images" (Deng et al., 2009a). When (Deng et al., 2009a) was released in 2009 it stated that the goal of ImageNet was to populate 80,000 subsets of images with 500 to 1,000 pictures. The ImageNet dataset is structured into subtrees and subsets, in an *IS A relationship*, meaning that if class A is a subset of class

---

<sup>1</sup>[https://github.com/simenvg/cloud\\_detection\\_framework](https://github.com/simenvg/cloud_detection_framework)

B, any object that satisfies class A's specification also satisfies class B's specification. For instance would fish be a subset of an animal. ImageNet contains more than 14 million images that have been annotated by ImageNet with the objects that are in the images. In more than one million of the images bounding boxes are also provided.

### Pascal, VOC2007 and VOC2012

Pascal delivers the datasets for the annual Visual Object Classes (VOC) Challenge. The datasets relevant for this report are VOC2007 and VOC2012. VOC2007 consists of 5,011 images with 12,608 objects in the training and validation set, while VOC2012 consists of 11,540 images with 27,540 objects. In figure 4.1 and 4.2 an overview of classes, images, and objects in VOC2007 and VOC2012 can be found. The combination of these two is denoted as VOC0712.

	train		val		trainval		test	
	img	obj	img	obj	img	obj	img	obj
Aeroplane	112	151	126	155	238	306	-	-
Bicycle	116	176	127	177	243	353	-	-
Bird	180	243	150	243	330	486	-	-
Boat	81	140	100	150	181	290	-	-
Bottle	139	253	105	252	244	505	-	-
Bus	97	115	89	114	186	229	-	-
Car	376	625	337	625	713	1250	-	-
Cat	163	186	174	190	337	376	-	-
Chair	224	400	221	398	445	798	-	-
Cow	69	136	72	123	141	259	-	-
Diningtable	97	103	103	112	200	215	-	-
Dog	203	253	218	257	421	510	-	-
Horse	139	182	148	180	287	362	-	-
Motorbike	120	167	125	172	245	339	-	-
Person	1025	2358	983	2332	2008	4690	-	-
Pottedplant	133	248	112	266	245	514	-	-
Sheep	48	130	48	127	96	257	-	-
Sofa	111	124	118	124	229	248	-	-
Train	127	145	134	152	261	297	-	-
Tvmonitor	128	166	128	158	256	324	-	-
Total	2501	6301	2510	6307	5011	12608	-	-

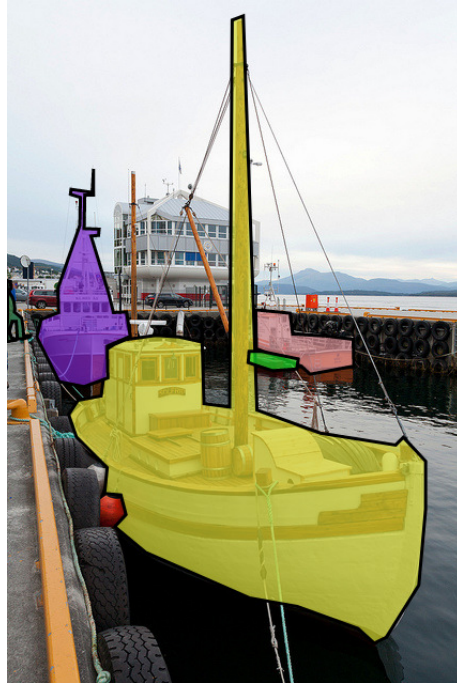
Figure 4.1: VOC2007 classes and images, image from (Everingham and Winn, 2007).

	train		val		trainval		test	
	img	obj	img	obj	img	obj	img	obj
Aeroplane	327	432	343	433	670	865	-	-
Bicycle	268	353	284	358	552	711	-	-
Bird	395	560	370	559	765	1119	-	-
Boat	260	426	248	424	508	850	-	-
Bottle	365	629	341	630	706	1259	-	-
Bus	213	292	208	301	421	593	-	-
Car	590	1013	571	1004	1161	2017	-	-
Cat	539	605	541	612	1080	1217	-	-
Chair	566	1178	553	1176	1119	2354	-	-
Cow	151	290	152	298	303	588	-	-
Diningtable	269	304	269	305	538	609	-	-
Dog	632	756	654	759	1286	1515	-	-
Horse	237	350	245	360	482	710	-	-
Motorbike	265	357	261	356	526	713	-	-
Person	1994	4194	2093	4372	4087	8566	-	-
Pottedplant	269	484	258	489	527	973	-	-
Sheep	171	400	154	413	325	813	-	-
Sofa	257	281	250	285	507	566	-	-
Train	273	313	271	315	544	628	-	-
Tvmonitor	290	392	285	392	575	784	-	-
Total	5717	13609	5823	13841	11540	27450	-	-

Figure 4.2: VOC2012 classes and images, image from (Everingham and Winn, 2012).

## COCO

In 2015 Microsoft released the COCO (Common Objects in Context) dataset, used for object recognition (Lin et al., 2014). The COCO dataset consists of 2.5 million labeled instances in 328,000 images, and are "images of complex everyday scenes containing common objects in their natural context" (Lin et al., 2014). Each object in the image is segmented and labeled separately, instead of semantically which is done in many other datasets. This means that objects of the same class are segmented into different instances. The COCO dataset has 3,146 images containing boats. An example image from the COCO dataset is shown in figure 4.3.



**Figure 4.3:** Example image from the COCO dataset, different objects are segmented at pixel values.

### 4.1.2 Custom datasets

A big part of the project has been to compile and annotate a dataset of images relevant to this project. This means that images are captured in environments where the algorithm should perform well, this being in areas around Trondheimsfjorden or similar areas.

Dataset	Number of images
Trondheim	114
Trondheimsfjorden	472
Hovik	1,197
Various	133
Total	1,916

These datasets were then divided into the following datasets divided into three different categories.

Category	Dataset	Number of images	Category total
Sailing boats	Trondheimsfjorden	472	1,723
	BoatsFar	1,215	
	BoatsClose	36	
Moored boats and buildings	MooredBoats	108	108
	MooredBoatsHouses	108	
Buildings	Houses	89	89

**Table 4.1:** In the dataset names houses are used for buildings to let the abbreviation "h" be used for buildings ([h]ouses), since "b" is used for boats later in this work. MooredBoats and MooredBoatsHouses contains the same images, in MooredBoatsHouses the buildings in the dataset are labeled as buildings, in MooredBoats they are not.

In (Tangstad, 2017), the detection algorithm used misclassified buildings as boats, and this issue has also been observed during testing of Yolo and SSD. For this reason, datasets consisting images of buildings were compiled, to be able to verify whether training on these images will reduce these misclassifications. The Trondheimsfjorden dataset has been kept as is, and not split into the other sub-datasets. The reasoning for this is that how the algorithm performs in this environment is an interesting performance metric in this project.

All the datasets have been split into test and train directories, where 75 % of the images are used in training, and 25 % are employed in testing. This is done randomly, but the same configuration is used in all training and testing to ensure that the detection algorithms have the same preconditions.

### Labeling of Custom Dataset

The custom dataset was annotated using LabelImg <sup>2</sup>. LabelImg lets the user choose the output format and simplifies the labeling process, also for multiclass-labeling. During labeling the VOC Annotation Guidelines (Everingham and Winn, 2012) have been used, which are:

- What to label
  - All objects of the defined categories, unless:
    - \* you are unsure what the object is.
    - \* the object is very small (at your discretion).
    - \* less than 10-20% of the object is visible, such that you cannot be sure what class it is. e.g., if only a tyre is visible it may belong to car or truck so cannot be labeled car, but feet/faces can only belong to a person.
  - If this is not possible because too many objects, mark the image as bad.
- Viewpoint

---

<sup>2</sup><https://github.com/tzutalin/labelImg>

- Record the viewpoint of the ‘bulk’ of the object, e.g. the body rather than the head. Allow viewpoints within 10-20 degrees. If ambiguous, leave as ‘Unspecified.’ Unusually rotated objects, e.g. upside-down people should be left as ‘Unspecified.’
- Bounding box
  - Mark the bounding box of the visible area of the object (not the estimated total extent of the object). Bounding box should contain all visible pixels, except where the bounding box would have to be made excessively large to include a few additional pixels (less than 5%) e.g., a car aerial.
- Truncation
  - If more than 15-20% of the object lies outside the bounding box mark as Truncated. The flag indicates that the bounding box does not cover the total extent of the object.
- Occlusion
  - If more than 5% of the object is occluded within the bounding box, mark as Occluded. The flag indicates that the object is not visible within the bounding box.
- Image quality/illumination
  - Images which are poor quality (e.g., excessive motion blur) should be marked bad. However, poor illumination (e.g., objects in silhouette) should not count as poor quality unless objects cannot be recognized. Images made up of multiple images (e.g., collages) should be marked bad.
- Clothing/mud/ snow etc.
  - If an object is ‘occluded’ by a close-fitting occluder e.g., clothing, mud, snow, etc., then the occluder should be treated as part of the object.
- Transparency
  - Do label objects visible through glass but treat reflections on the glass as occlusion.
- Mirrors
  - Do label objects in mirrors.
- Pictures
  - Label objects in pictures/posters/signs only if they are photorealistic but not if cartoons, symbols, etc.

## 4.2 Detection framework

Object detection using deep learning based methods is a field of research which is rapidly changing, and new algorithms and methods have continuously been presented in recent years. When new methods and articles are released to the public, these algorithms have performance data which makes it possible to compare them to already existing algorithms. These statistics are based on the performance of the algorithms on standard datasets, such as COCO (Lin et al.), VOC Pascal (Everingham and Winn, 2007), (Everingham and Winn, 2012), and Imagenet (Deng et al., 2009b), and gives an estimate of how fast and how accurate the algorithm is. When trained on these datasets, which include multiple classes, e.g. persons, cars and dogs, they become general purpose object detectors. In this project, the object of interest is maritime vessels, and possibly other objects related to navigation and obstacle avoidance. Thus, the focus in this report is a specialized object detector that is trained for detecting these objects. The performance statistics from object detection articles can thus only be used as a guideline for how well it will perform as a specialized object detector in a maritime environment.

The different, currently existing, object detection algorithms could potentially perform on a vastly different level depending e.g. on the lighting conditions, how big the object of interest is in the image, and which data they are trained on. Their performance can also significantly differ based on how the algorithm is configured. Yolo, for instance, can perform better on smaller objects by increasing the grid size mentioned in chapter 3.5. The performance of different detection algorithms in maritime environments based on their configuration and on which data it is trained on, is a landscape that is not explored fully. This project aims to make it possible to more efficiently and systematically navigate in this landscape and to begin a heuristic approach to find directions worth continuing to research.

```

simenvg@instance-yolo-build: ~/cloud_detection_framework/yolo_119x31
96 route 85 61
97 conv 256 1 x 1 / 1 38 x 38 x 768 -> 38 x 38 x 256 0.568 BFL0Ps
98 conv 512 3 x 3 / 1 38 x 38 x 256 -> 38 x 38 x 512 3.407 BFL0Ps
99 conv 256 1 x 1 / 1 38 x 38 x 512 -> 38 x 38 x 256 0.379 BFL0Ps
100 conv 512 3 x 3 / 1 38 x 38 x 256 -> 38 x 38 x 512 3.407 BFL0Ps
101 conv 256 1 x 1 / 1 38 x 38 x 512 -> 38 x 38 x 256 0.379 BFL0Ps
102 conv 512 3 x 3 / 1 38 x 38 x 256 -> 38 x 38 x 512 3.407 BFL0Ps
103 conv 21 1 x 1 / 1 38 x 38 x 512 -> 38 x 38 x 21 0.031 BFL0Ps
104 yolo
105 route 91
106 conv 128 1 x 1 / 1 38 x 38 x 256 -> 38 x 38 x 128 0.095 BFL0Ps
107 upsample 2x 38 x 38 x 128 -> 76 x 76 x 128
108 route 97 36
109 conv 128 1 x 1 / 1 76 x 76 x 384 -> 76 x 76 x 128 0.568 BFL0Ps
110 conv 256 3 x 3 / 1 76 x 76 x 128 -> 76 x 76 x 256 3.407 BFL0Ps
111 conv 128 1 x 1 / 1 76 x 76 x 256 -> 76 x 76 x 128 0.379 BFL0Ps
112 conv 256 3 x 3 / 1 76 x 76 x 128 -> 76 x 76 x 256 3.407 BFL0Ps
113 conv 128 1 x 1 / 1 76 x 76 x 256 -> 76 x 76 x 128 0.379 BFL0Ps
114 conv 256 3 x 3 / 1 76 x 76 x 128 -> 76 x 76 x 256 3.407 BFL0Ps
115 conv 21 1 x 1 / 1 76 x 76 x 256 -> 76 x 76 x 21 0.062 BFL0Ps
116 yolo
Loading weights from /home/simenvg/data/model/backup/yolo_obj_train_4800.weights...Done!
[ 0 ] boats_far
[ 1 ] trondheimsfjorden_video
[ 2 ] boats_close
[ 3 ] buildings
[ 4 ] buildings_boats
[ 5 ] video_kams
[ 6 ] trondheimsfjorden
[ 7 ] buildings_boats_buildings_not_labelled
Input the number for the datasets you wish to test on, separate numbers with space: █

```

**Figure 4.4:** Example image from the Cloud Detection Framework, here the user is prompted which datasets to test Yolo on.

### 4.2.1 Hardware

One of the first issues one encounters while trying to train a convolutional neural network is the need for specific hardware that has the computational capacity to train and test the network in a reasonable amount of time. All the object detection algorithms mentioned in this thesis are implemented with support for Graphics Processing Unit (GPU) computations. According to Nvidia, GPUs alone increased the computational speed by a factor of 1,000 over a span of 10 years, greatly supporting the progress in deep learning (Dettmers, 2015). The most popular library used for GPU processing for neural networks is today CUDA, which is developed by Nvidia. Hence, the GPU needed for more efficient training and testing of neural networks has to be a Nvidia GPU.

Another problem that arises when working with the cutting edge detection algorithms is to get an environment set up correctly. Libraries, programs, and drivers have to be installed the correct way, and configuration files and paths must be set accordingly. This may sound trivial, but the road from an annotated dataset to a trained object detector can be long and frustrating. A goal in this project is to make this process as simple as possible, this enables future master students or researchers to focus more on the actual problem and not spend time on creating the system.

### 4.2.2 Google Cloud

The solution to the hardware problem in this project is to use a cloud computing service, in this case, Google Cloud. Other cloud computing solutions exist, where some of the most popular are Amazon AWS and Microsoft Azure. Google Cloud was chosen because it is an economical choice in a testing phase.

Google Cloud lets you open an instance where you can specify the operating system and hardware according to the requirements of the user. The instance opens on a clean installation of the selected operating system, and an object detection environment can, therefore, be set up the same way for every instance. In this project, the operating system used is Ubuntu 16.04 and the GPU used is a Nvidia Tesla K80. A detailed explanation of how Google Cloud was configured can be seen in appendix A.

It should also be noted that this system work on other cloud computing services, as long as the hardware satisfies the requirements.

### 4.2.3 Cloud Instance Setup

The Google Cloud instance can be configured by cloning the Cloud Detection Framework Github repository<sup>3</sup>. Running the `setup.sh` file will install and set up the following

- CUDA and cudnn
  - Nvidia libraries used to run computations on the GPU.
- Tensorflow
  - Google’s software library for dataflow programming

---

<sup>3</sup>[https://github.com/simenvg/cloud\\_detection\\_framework](https://github.com/simenvg/cloud_detection_framework)

- Tensorflows models directory from GitHub is also downloaded and with it its object detection API
- Yolo
  - Yolo’s GitHub repository is downloaded, modified to have GPU support and built
- Directory structure
  - Directories and subdirectories are made to handle training data input and to have a consistent system configuration for the different object detection algorithms
- Required Python libraries
- Virtual Environments
  - A python3 virtual environment is set up to handle compatibility with different frameworks.

When the setup file has been executed, data can be uploaded to the Google cloud instance. When data have been uploaded, and the Google cloud instance has been set up by running `setup.sh`, the system is ready to train.

#### 4.2.4 Training Yolo

In this project Yolov3 is used, the latest version of Yolo at the time of writing. To train Yolo, run the file `train_yolo.py`, this will:

- Prompt the user and ask which datasets the user wishes to train on.
- Set up a model folder where all files needed for testing is saved.
- Convert annotation files for the images to Yolo format
- Make a `train.txt` file which contains the path to all the images that are to be used in training
- Make configuration files based on the training data needed to run YOLO.
- Edit Yolo’s main configuration file (`.cfg` file) to have the correct batch-size, the number of classes, subdivision, and filters, based on the number of classes in the training data and the hardware used on the Google cloud instance.
- Load pre-trained weights
- Begin training.

During training, Yolo will save weights every 400 iterations to the model repository. It will continue training until the user exits the process, or when it reaches 40,000 iterations. When the process runs the average loss will be written to the terminal for every iteration, and training can be stopped when it is sufficiently low. In this project all the detection models were trained for approximately 24 hours.



**Pre-trained weights yolo**

Yolo comes with the following pre-trained weights

Model	Top-1	Top-5	Ops	GPU	CPU	Weights
AlexNet	57.0	80.3	2.27 Bn	3.1 ms	0.29 s	238 MB
Darknet Reference	61.1	83.0	0.96 Bn	2.9 ms	0.14 s	28 MB
VGG-16	70.5	90.0	30.94 Bn	9.4 ms	4.36 s	528 MB
Extraction	72.5	90.8	8.52 Bn	4.8 ms	0.97 s	90 MB
Darknet19	72.9	91.2	7.29 Bn	6.2 ms	0.87 s	80 MB
Darknet19 448x448	76.4	93.5	22.33 Bn	11.0 ms	2.96 s	80 MB
Resnet 18	70.7	89.9	4.69 Bn	4.6 ms	0.57 s	44 MB
Resnet 34	72.4	91.1	9.52 Bn	7.1 ms	1.11 s	83 MB
Resnet 50	75.8	92.9	9.74 Bn	11.4 ms	1.13 s	87 MB
Resnet 101	77.1	93.7	19.70 Bn	20.0 ms	2.23 s	160 MB
Resnet 152	77.6	93.8	29.39 Bn	28.6 ms	3.31 s	220 MB
ResNeXt 50	77.8	94.2	10.11 Bn	24.2 ms	1.20 s	220 MB
ResNeXt 101 (32x4d)	77.7	94.1	18.92 Bn	58.7 ms	2.24 s	159 MB
ResNeXt 152 (32x4d)	77.6	94.1	28.20 Bn	73.8 ms	3.31 s	217 MB
Densenet 201	77.0	93.7	10.85 Bn	32.6 ms	1.38 s	66 MB
Darknet53	77.2	93.8	18.57 Bn	13.7 ms	2.11 s	159 MB
<b>Darknet53 448x448</b>	<b>78.5</b>	<b>94.7</b>	<b>56.87 Bn</b>	<b>26.3 ms</b>	<b>7.21 s</b>	<b>159 MB</b>

**Table 4.2:** These models are trained on ImageNet, and the top-1 and top-5 columns show the accuracy of the pre-trained model on the ImageNet dataset. In this project, the Darknet53 448x448 model has been used. Table from (Redmon and Farhadi, 2018).

## 4.2.5 Training SSD

In this project Tensorflows implementation of SSD Mobilenet is used. To train SSD run the file `train_ssd.py`, this will:

- Prompt the user and ask which datasets the user wishes to train on
- Convert annotations and images to tfrecords. Tfrecords are Tensorflows binary file format. Binary files uses less storage and are beneficial when working with large datasets.
- Update SSD's configuration file with correct paths to training data, classes, and filters.
- Load pre-trained weights
- Begin training

`train_ssd.py` will also save all files needed for testing to the model directory. It will save the weights based on time, and not based on iterations like Yolo. At what iteration the happens depends on the size of the dataset and the hardware used.

### Pre-trained weights SSD

SSD is implemented using Tensorflow's object detection API. Tensorflow offers several detection algorithms shown in table 4.3. These are trained on the Coco dataset and comes with different configurations. The cloud detection framework is designed to be able to implement other detection algorithms easily. The SSD version chosen in this project is `ssd_mobilenet_v1_coco`. The Cloud Detection Framework is designed such that to implement any of the other models shown in figure 4.3 one only needs to download another model and edit a few lines in a configuration file. The models implemented in Tensorflow's object detection API is also rapidly expanding. In the summer of 2017, there were eight models available, in December of 2018, there are 25. This means that by using the Cloud Detection Framework one can easily train and test new, cutting-edge models that Google supply through Tensorflow.

Model name	Speed (ms)	COCO mAP[ <sup>1</sup> ]
<b>ssd_mobilenet_v1_coco</b>	<b>30</b>	<b>21</b>
ssd_mobilenet_v1_0.75_depth_coco	26	18
ssd_mobilenet_v1_quantized_coco	29	18
ssd_mobilenet_v1_0.75_depth_quantized_coco	29	16
ssd_mobilenet_v1_ppn_coco	26	20
ssd_mobilenet_v1_fpn_coco	56	32
ssd_resnet_50_fpn_coco	76	35
ssd_mobilenet_v2_coco	31	22
ssd_mobilenet_v2_quantized_coco	29	22
ssdlite_mobilenet_v2_coco	27	22
ssd_inception_v2_coco	42	24
faster_rcnn_inception_v2_coco	58	28
faster_rcnn_resnet50_coco	89	30
faster_rcnn_resnet50_lowproposals_coco	64	
rfcn_resnet101_coco	92	30
faster_rcnn_resnet101_coco	106	32
faster_rcnn_resnet101_lowproposals_coco	82	
faster_rcnn_inception_resnet_v2_atrous_coco	620	37
faster_rcnn_inception_resnet_v2_atrous_lowproposals_coco	241	
faster_rcnn_nas	1833	43
faster_rcnn_nas_lowproposals_coco	540	
mask_rcnn_inception_resnet_v2_atrous_coco	771	36
mask_rcnn_inception_v2_coco	79	25
mask_rcnn_resnet101_atrous_coco	470	33
mask_rcnn_resnet50_atrous_coco	343	29

**Table 4.3:** Pre-trained models SSD.

## 4.2.6 Testing with Cloud Detection Framework

During testing of the detection algorithms, a problem arose when testing on large datasets. When testing on hundreds of images containing several objects the data could not be saved in JSON format as a txt-file due to heavy memory requirements. Thus, a different solution was implemented where detections were saved in a database using SQL. This divides the process of testing into two phases. One where the algorithm is run on the test images and all detections are written to a database. The other phase draws the detections, and the ground truth bounding boxes onto the test images and saves them, and calculates statistics on how well the algorithm performed. These results are saved to the "results" directory.

Both Yolo and SSD will assume that the model is set up as it is after training. This means that a trained model can be downloaded, and uploaded to new instances in Google Cloud. This makes it easy to reuse trained models, and to continue working on other's research.

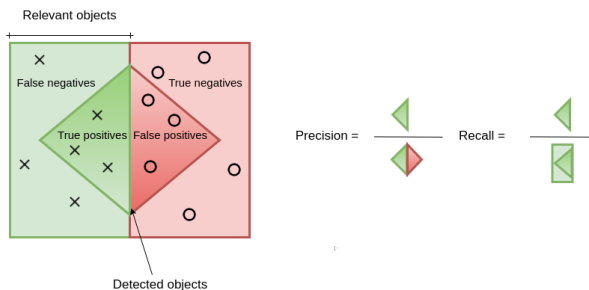


# Evaluation and Testing Methodology

This chapter covers the methodology used for evaluating the detection algorithms used, as well as the dataset used for testing. The goal of this chapter is to show how the two detection algorithms have been tested, and how they can be compared using the performance metrics defined in this chapter.

## 5.1 Precision and Recall

Two common performance measures used in object detection are *precision* and *recall*. By using these two measures, both the false negatives and the false positives are taken into account. Recall measures how many of the true positives in the image are detected, and precision measures how many of the detections that are correct. This is also shown graphically in figure 5.1. In detection, theory recall is often referred to as *probability of detection*. A high precision would correspond to what detection theory calls a low false alarm rate.



**Figure 5.1:** Precision and recall, precision is the number of correct detections (true positives) divided by the total amount of detections. Recall is the amount of correct detections divided by the number of ground truth objects.

## 5.2 Bounding Box Evaluation

To be able to calculate the precision and recall, there must be a definition of what a correct detection is. In (Everingham et al., 2010) this is defined the following way: "Detections are considered true or false positives based on the area of overlap with ground truth bounding boxes. To be considered a correct detection, the area of overlap  $a_0$  between the predicted bounding box  $B_p$  and ground truth bounding box  $B_{gt}$  must exceed 50% by the formula:

$$a_0 = \frac{\text{area}(B_p \cap B_{gt})}{\text{area}(B_p \cup B_{gt})}$$

The area of overlap is the same as intersect over union explained in figure 3.3. The same definition is used in this report.

## 5.3 Precision/recall Curves

In the VOC challenge the detection task is judged by a precision/recall curve (Everingham et al., 2010). The precision/recall curve is made by checking the precision and recall for the detection algorithm at different confidence score thresholds. The confidence score used is the score returned from the detection algorithms along with the bounding box, i.e., how confident the algorithm is in the bounding box. If the confidence score threshold is low, the detection algorithm will likely detect more objects of interest, but also identify more false positives. This means that for a low confidence score threshold the recall will be high and the precision low. In figure 5.2 an example of a precision/recall curve is shown, and the results in this report will be presented in the same manner.

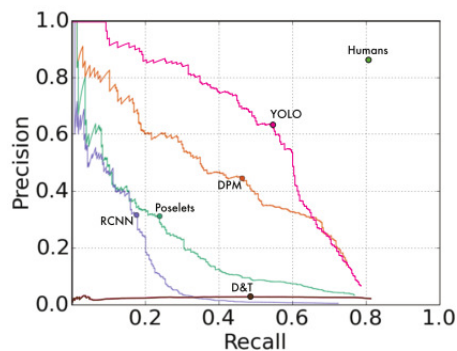


Figure 5.2: Precision/recall curve from (Redmon et al., 2016)

We want both the recall and the precision to be as high as possible, meaning that in the precision/recall curve we would like to get as far up in the top right corner as possible. As shown in figure 5.2, the precision decreases while the recall increases. This is the expected behavior since more objects are detected when the threshold based on the confidence score is lowered. More detections also increase the number of false positives. Sometimes it can be hard to decide precisely what the optimal confidence score threshold should be, since this is a trade-off between recall and precision and will be discussed further in the next chapter.

## 5.4 Average precision

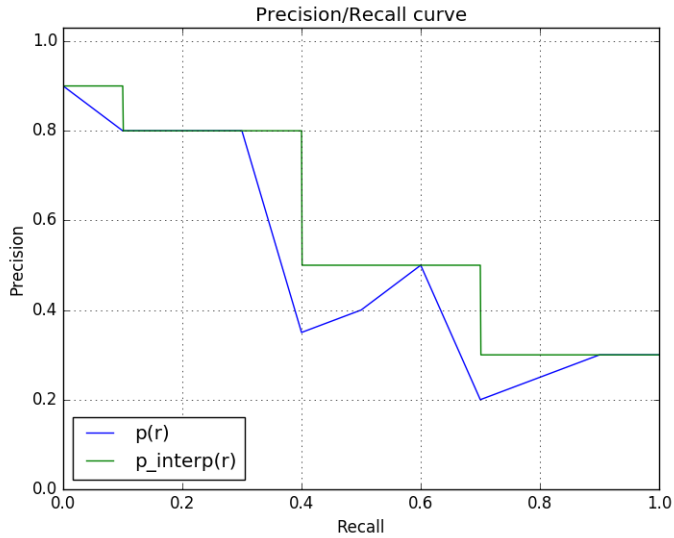
Average precision is a commonly used metric in object detection and is used in the papers of, e.g. Faster R-CNN, YOLO, and SSD. It is the average of the maximum precision at different recall levels.

$$AP = \frac{1}{11} \sum_{r \in \{0.0, \dots, 1.0\}} p_{interp}(r) \quad (5.1)$$

Where

$$p_{interp}(r) = \max_{\tilde{r} \geq r} p(\tilde{r}) \quad (5.2)$$

This means that for each recall value in 0.0, ... , 1.0 the highest precision level at that recall level, or a higher recall level, is added to the sum. This is illustrated in figure 5.3, where the green line shows  $p_{interp}(r)$



**Figure 5.3:**  $p(r)$  and  $p_{interp}(r)$ .

## 5.5 Confusion Matrix

A confusion matrix is another way of presenting the performance of a classification algorithm. The confusion matrix gives a more intuitive representation of the results, and gives a good indication of how well the detection algorithm would work in a real scenario. In this project the goal is to identify boats, thus it is a two-class problem with boat and not-boat as the two classes.

Figure 5.4 shows how the confusion matrix will be used in this project. True positives are boats detected correctly, false positives are detections that are incorrect, false negatives are undetected boats. The detections algorithms developed in this project does not seek to identify non-boat objects, rather to overlook them, thus true negatives cannot be quantized and are not relevant in this work.

		Actual class	
		Boat	Non-boat
Predicted class	Boat	True Positives	False Positives
	Non-boat	False Negatives	True Negatives

**Figure 5.4:** Confusion matrix.



# Chapter 6

## Results

This chapter contains results from testing six different detection models on four test sets. The six models consists of three SSD models and three Yolo models trained on different data. This produces a significant amount of results, and makes the task of presenting the results in an orderly fashion challenging. To try to overcome this challenge this chapter is structured in the following way:

1. Present training sets and models. Explain abbreviations used later in the chapter.
2. Overview of results.
3. Present case studies covering more interesting aspects of results.

The case studies and test sets were chosen to shed some light on the following research questions that should be addressed.

- How does training on a building class affect the performance on detection of boats?
- How does training on boats in coast near environments affect detection accuracy in open sea?
- How does SSD compare to Yolo when trained and tested on the same data?
- In which situations do the detection algorithms struggle, and in which situations is the performance satisfactory?

In this chapter, datasets are mentioned when referring both to training and testing of models. All the custom datasets are split into test and training subsets, which are used for testing and training, respectively.

The terms "test set" and "training set" are used widely in this chapter. A test set or a training set is made up of combinations of one or more datasets.

## 6.1 Training Sets and Detection Models

Yolo and SSD were trained on three different training sets. The names of the training sets are given in table 6.1 along with the datasets each training set is composed of, and their abbreviation.

Training set	Datasets
Boats Sailing (BS)	Trondheimsfjorden BoatsClose BoatsFar
Boats Sailing and Moored boats (BSM)	Trondheimsfjorden BoatsClose BoatsFar MooredBoats
Boats Sailing, Moored boats and Houses (BSMH)	Trondheimsfjorden BoatsClose BoatsFar MooredBoatsHouses Houses

**Table 6.1:** Training sets used to train Yolo and SSD models.

These training sets were used to train three SSD models and three Yolo models. Henceforth these models will be denoted with the training set abbreviation as subscript:  $Yolo_{BS}$ ,  $SSD_{BS}$ ,  $Yolo_{BSM}$ ,  $SSD_{BSM}$ ,  $Yolo_{BSMH}$  and  $SSD_{BSMH}$ .

## 6.2 Results Overview

To present an overview of the results two performance metrics have been used. Average precision is presented in chapter 6.2.1, and confusion matrices in chapter 6.2.2. Chapter 6.2.3 discusses the most important aspects of these results.

### 6.2.1 Average Precision

The average precision of the models on the different test sets is shown in table 6.2.

Test set	$Yolo_{BS}$	$SSD_{BS}$	$Yolo_{BSM}$	$SSD_{BSM}$	$Yolo_{BSMH}$	$SSD_{BSMH}$
MooredBoats	0.182	0.140	0.707	0.515	0.701	0.586
Trondheimsfjorden	0.885	0.860	0.900	0.750	0.908	0.876
BoatsClose and BoatsFar	0.509	0.278	0.533	0.239	0.647	0.292
BoatsClose, BoatsFar and Trondheimsfjorden	0.577	0.375	0.600	0.329	0.726	0.391

**Table 6.2:** Average precisions

## 6.2.2 Confusion Matrices

As explained in chapter 5.5, confusion matrices give a more intuitive understanding of the results than average precisions. In this chapter two tables are presented, table 6.4 where the confidence threshold is set to 0.25, and 6.5 with a confidence threshold of 0.5.

The bounding boxes Yolo and SSD outputs comes with a confidence score, indicating how confident the algorithm is in the detection. A higher confidence threshold could lead to fewer false positives, but possibly also fewer true positives. Only detections with a higher confidence score than the confidence thresholds set in table 6.4 and table 6.5 are accounted for in the tables.

Table 6.4 and 6.5 are composed of the confusion matrices for all six detection models on the four test sets. Each cell in the table is the confusion matrix for the corresponding detection model and test set. Table 6.3 explains what the numbers in each cell represent.

Model	Test set(s) (Number of objects in test set)	
	True positives	False positives
	False negatives	

**Table 6.3:** Confusion matrix explanation

	Moored-Boats(58)		Tr.fjorden(174)		BoatsClose, BoatsFar(818)		BoatsClose, BoatsFar, Tr.fjorden(992)	
Yolo <sub>BS</sub>	13	21	164	22	551	435	715	457
	45		10		267		277	
Yolo <sub>BSM</sub>	39	4	167	8	555	263	722	271
	19		7		263		270	
Yolo <sub>BSMH</sub>	40	6	173	3	611	214	784	217
	18		1		207		208	
SSD <sub>BS</sub>	7	15	151	17	359	315	510	332
	51		23		459		482	
SSD <sub>BSM</sub>	32	20	146	34	323	437	469	471
	26		28		495		523	
SSD <sub>BSMH</sub>	32	13	150	23	365	282	515	305
	26		24		453		477	

**Table 6.4:** Confusion matrix with the confidence threshold set to 0.25. Explanation of cells in the table can be found in table 6.3.

	Moored-Boats(58)		Tr.fjorden(174)		BoatsClose, BoatsFar(818)		BoatsClose, BoatsFar, Tr.fjorden(992)	
Yolo <sub>BS</sub>	11	10	160	15	524	311	684	326
	47		14		294		308	
Yolo <sub>B<sub>SM</sub></sub>	35	1	165	6	498	198	663	204
	23		9		320		329	
Yolo <sub>B<sub>SMH</sub></sub>	36	4	170	2	577	149	747	151
	22		4		241		245	
SSD <sub>BS</sub>	6	11	142	11	333	251	475	262
	52		32		485		517	
SSD <sub>B<sub>SM</sub></sub>	29	9	136	22	299	338	435	360
	29		38		519		557	
SSD <sub>B<sub>SMH</sub></sub>	31	6	147	14	319	214	466	228
	27		27		499		526	

**Table 6.5:** Confusion matrix with the confidence threshold set to 0.5. Explanation of cells in the table can be found in table 6.3.

### 6.2.3 Main results

#### Best results

Yolo<sub>B<sub>SMH</sub></sub> has the best overall results. On the Trondheimsfjorden test set Yolo<sub>B<sub>SMH</sub></sub> has an average precision of 0.908, which is impressive. Table 6.4, where the confidence threshold is set to 0.25, shows that Yolo<sub>B<sub>SMH</sub></sub> detects 173 out of 174 boats in this test set, with only three false positives. When the confidence threshold is set to 0.5, Yolo<sub>B<sub>SMH</sub></sub> detects 170 out of 174 boats, with two false positives, as shown in table 6.5. Even though the results on the other test sets are less impressive than on the Trondheimsfjorden test set, they are not bad. In (Redmon and Farhadi, 2018), Yolo gets a mean average precision, which is average precision for multi-class detection, of 0.553 on the COCO dataset. To compare the average precision on the custom datasets to the results on the multi-class detection problem on the COCO dataset is not a fair comparison, but it still is worth noting to give a better intuition for what the average precisions shown in table 6.2 means.

#### Effect of training differences

The hypothesis before running the tests in this work was that more relevant training data used in a model would mean better test results. This means that the expected results was that Yolo<sub>B<sub>SMH</sub></sub> would be better than Yolo<sub>B<sub>SM</sub></sub> which again would be better than Yolo<sub>BS</sub>, and the same for the SSD models. Table 6.2 shows this hypothesis to be correct except for SSD<sub>BS</sub> and SSD<sub>B<sub>SM</sub></sub>, where SSD<sub>BS</sub> performs better than SSD<sub>B<sub>SM</sub></sub>. Yolo<sub>B<sub>SM</sub></sub> is only slightly better than Yolo<sub>BS</sub> on all datasets except the MooredBoats test set. This means that, since Yolo<sub>B<sub>SMH</sub></sub> and SSD<sub>B<sub>SMH</sub></sub> have the best results, training on a building class has a significant effect on the test results on all test sets. This will be further discussed in chapter 6.3. SSD<sub>BS</sub> and Yolo<sub>BS</sub> detects the least boats on the MooredBoats test set, this is expected since these are the only models that have not been trained on data similar to this test set.

**Yolo vs. SSD**

As shown in the tables in chapter 6.2.1 and 6.2.2, Yolo performs better than SSD in this project. One of the critical differences between Yolo and SSD was discussed in chapter 3.7, where SSD uses grids at multiple scales to detect objects, while Yolo only applies one. The hypothesis was that this would make SSD better at identifying boats that varied more in size, while Yolo would only detect boats within a certain size range. This hypothesis was not confirmed during the work on this project. Yolo detects more boats and has fewer misclassifications than SSD in all the tests done in this work. Still, it should be noted that the variations in size of boats used in this thesis, could have been too small for the effect to be seen.

In this project SSD Mobilenet was used, a lightweight version of SSD meant for real-time solutions. SSD Mobilenet is faster, but less accurate compared to the original implementation. The reason for choosing Mobilenet over the original implementation was that it was used in a project at DNV GL the summer of 2018 with promising results. While Yolo and the original implementation of SSD has comparable accuracy, the accuracy of SSD Mobilenet is lower, meaning Yolo's superior accuracy is not entirely unexpected.

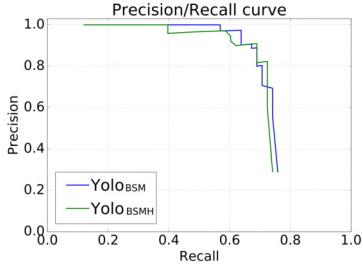
### 6.3 Case Study 1: Effect from training on buildings

Both (Tangstad, 2017) and (Kamsvåg, 2018) implemented an object detector for maritime vessels using Faster R-CNN. A challenge highlighted in both these projects was the misclassification of waterfront buildings as boats. To address this problem the datasets MooredBoatsHouses and Houses were produced. By training the object detector on a building class, it may be able to detect the buildings as buildings and thus suppress the classifications of waterfront buildings as boats. As a first step in trying to verify this hypothesis, both Yolo and SSD were trained on two different sets of data. The two datasets they are trained on contains the same set of images with two exceptions:

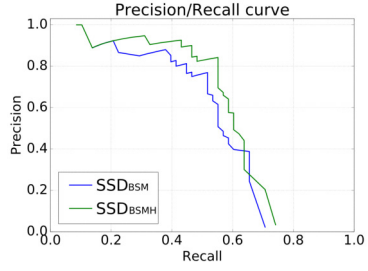
1. For  $\text{Yolo}_{\text{BSM}}$  and  $\text{SSD}_{\text{BSM}}$  only boats are labeled.
2. For  $\text{Yolo}_{\text{BSMH}}$  and  $\text{SSD}_{\text{BSMH}}$  buildings are labeled in the dataset MooredBoatsHouses, and a dataset containing only waterfront buildings was added to the training set.

Thus, all the models have been trained on the same boat objects, while  $\text{Yolo}_{\text{BSMH}}$  and  $\text{SSD}_{\text{BSMH}}$  also have been trained on a building class. This has been done to try to minimize the variation in the variables that affect boat detection, and try to only look at how adding a building class affects the boat detection performance.

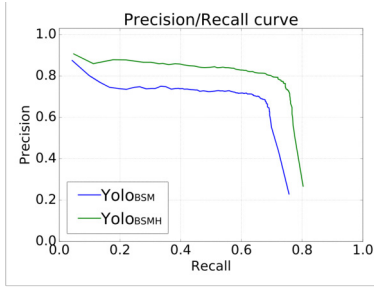
In figure 6.1 the precision/recall curves for the models are displayed for the four test cases. For both SSD and Yolo the model trained on buildings generally performs better than  $\text{Yolo}_{\text{BSM}}$  and  $\text{SSD}_{\text{BSM}}$ . However, the difference is more significant for SSD than for Yolo.



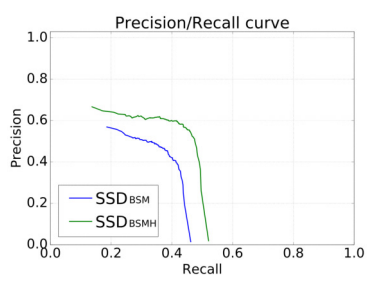
(a) Yolo tested on MooredBoats



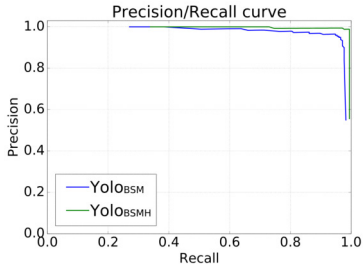
(b) SSD tested on MooredBoats



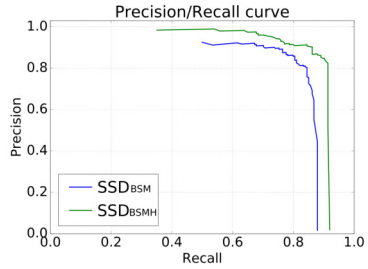
(c) Yolo tested on BoatsClose and BoatsFar



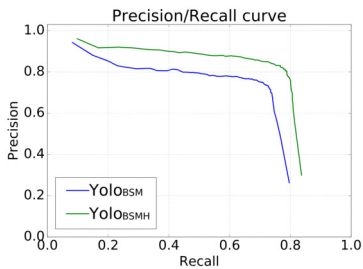
(d) SSD tested on BoatsClose and BoatsFar



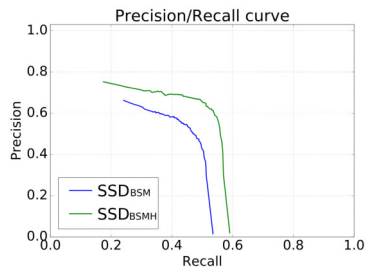
(e) Yolo tested on Trondheimsfjorden



(f) SSD tested on Trondheimsfjorden



(g) Yolo tested on BoatsClose, BoatsFar and Trondheimsfjorden



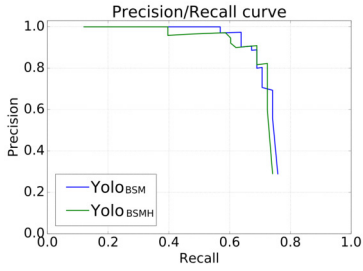
(h) SSD tested on BoatsClose, BoatsFar and Trondheimsfjorden

**Figure 6.1:** Precision/recall curves of Yolo<sub>BSM</sub>, Yolo<sub>BSMH</sub>, SSD<sub>BSM</sub> and SSD<sub>BSMH</sub>

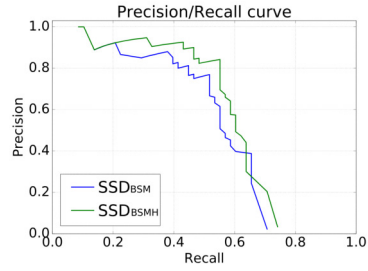
### 6.3.1 Tested on MooredBoats

The hypothesis before running the experiment was that the MooredBoats test set would be the most positively affected by training on a building class. Since MooredBoats consists of mostly moored boats, where there are images that contain buildings close to sea level one could suspect could be mistaken as boats, as shown in figure 6.2. The building in the bottom left corner is close to sea level, while also being relatively close in size to the boats compared to the other buildings in the image, which makes it a good candidate for a misclassification. However, neither  $Yolo_{BSM}$  nor  $SSD_{BSM}$  classifies this building as a boat. All the detectors,  $Yolo_{BSM}$ ,  $Yolo_{BSMH}$ ,  $SSD_{BSM}$  and  $SSD_{BSMH}$  detects all the boats in the image. Both  $SSD_{BSM}$  and  $SSD_{BSMH}$  detects the rightmost boat two times, and the bounding box differences between  $Yolo_{BSM}$  and  $Yolo_{BSMH}$  and between  $SSD_{BSM}$  and  $SSD_{BSMH}$  are so insignificant that training on the building class does not seem to help improve performance on this particular image. Both  $Yolo_{BSMH}$  and  $SSD_{BSMH}$  detects the buildings in the image well, as can be seen in figure 6.2e and 6.2f.

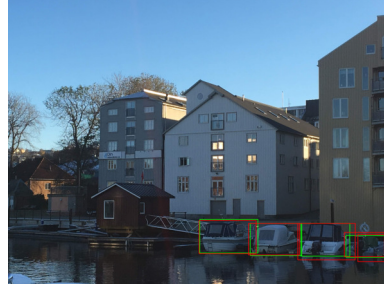
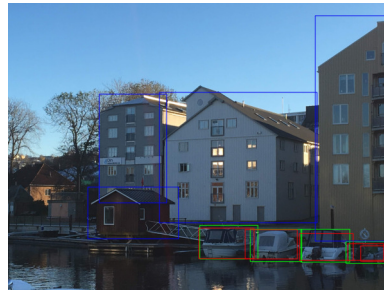




(a) Yolo tested on MooredBoats



(b) SSD tested on MooredBoats

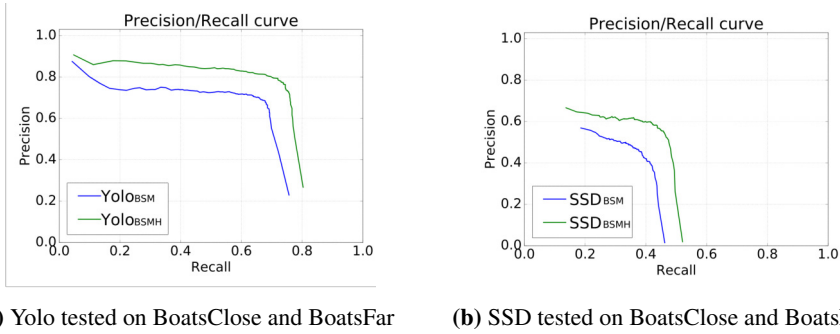
(c) Yolo<sub>BSM</sub>(d) SSD<sub>BSM</sub>(e) Yolo<sub>BSMH</sub>(f) SSD<sub>BSMH</sub>

**Figure 6.2:** Yolo<sub>BSM</sub>, Yolo<sub>BSMH</sub>, SSD<sub>BSM</sub>, SSD<sub>BSMH</sub> example image from MooredBoats test set. Green bounding boxes are ground truth, red bounding boxes are detected boats, blue bounding boxes are detected buildings.

### 6.3.2 Tested on the BoatsClose and BoatsFar

The datasets BoatsClose and BoatsFar consists mostly of images taken towards open sea, of boats that are sailing, i.e., the boats are not moored. It is on this test set the performance differences between Yolo<sub>BSM</sub> and Yolo<sub>BSMH</sub> and between SSD<sub>BSM</sub> and SSD<sub>BSMH</sub> differ the most, as can be seen in figure 6.3.

By going through the approximately 300 test images, some differences between the models become apparent. The differences between SSD<sub>BSM</sub> and SSD<sub>BSMH</sub> are clearer than the differences between Yolo<sub>BSM</sub> and Yolo<sub>BSMH</sub>.



**Figure 6.3:** Yolo<sub>B<sub>SM</sub></sub>, Yolo<sub>B<sub>SMH</sub></sub>, SSD<sub>B<sub>SM</sub></sub>, SSD<sub>B<sub>SMH</sub></sub> example image from MooredBoats test set. Green bounding boxes are ground truth, red bounding boxes are detected boats, blue bounding boxes are detected buildings

### SSD<sub>B<sub>SM</sub></sub> and SSD<sub>B<sub>SMH</sub></sub> on BoatsClose and BoatsFar

In around 10 percent of the test images, SSD<sub>B<sub>SM</sub></sub> detects land as a boat, in none of these images does SSD<sub>B<sub>SMH</sub></sub> perform the same way. An example of this is shown in figure 6.4. More examples of this behaviour can be found in appendix C.1



**Figure 6.4:** SSD<sub>B<sub>SM</sub></sub> detects land as boat. Red bounding boxes are boat detections, green bounding boxes are ground truth.

There are also some examples where SSD<sub>B<sub>SM</sub></sub> misclassifies buildings as boats, while SSD<sub>B<sub>SMH</sub></sub> does not. One example is shown in figure 6.5.



**Figure 6.5:** Leftmost red bounding box in  $SSD_{BSM}$  is a detection of a building as a boat, this is not detected as a boat in  $SSD_{BSMH}$ . Red bounding boxes are boat detections, green bounding boxes are ground truth.

### $Yolo_{BSM}$ and $Yolo_{BSMH}$ on BoatsClose and BoatsFar

As mentioned,  $Yolo_{BSM}$  and  $Yolo_{BSMH}$  do not differ as clearly as  $SSD_{BSM}$  and  $SSD_{BSMH}$ . There are examples of  $Yolo_{BSM}$  performing better than  $Yolo_{BSMH}$  and vice versa. In many of the misclassifications in the test data both  $Yolo_{BSM}$  and  $Yolo_{BSMH}$  mistakes the same object as a boat. By analyzing the results from both  $Yolo_{BSM}$  and  $Yolo_{BSMH}$ , the following tendencies can be seen:

- $Yolo_{BSM}$  and  $Yolo_{BSMH}$  have the disposition to wrongly classify the same objects as boats. See appendix C.2.4 for example images.
- $Yolo_{BSM}$  performs better in some cases, while  $Yolo_{BSMH}$  performs better in other. It is not clear what makes one better than the other in each specific case. See appendix C.2.2 and C.2.3 for example images
- $Yolo_{BSM}$  makes some misclassifications that could imply that training on a building class makes  $Yolo_{BSMH}$  more robust to the more incomprehensible misclassifications. See appendix C.2.1 for example images.

As opposed to the differences between  $SSD_{BSM}$  and  $SSD_{BSMH}$ , the differences in the results between  $Yolo_{BSM}$  and  $Yolo_{BSMH}$  are hard to pinpoint. Both  $Yolo_{BSM}$  and  $Yolo_{BSMH}$  make misclassifications, but  $Yolo_{BSM}$  seems to make slightly more than  $Yolo_{BSMH}$ .  $Yolo_{BSMH}$  also avoids making some misclassifications  $Yolo_{BSM}$  makes in the background, as shown in figure 6.6.

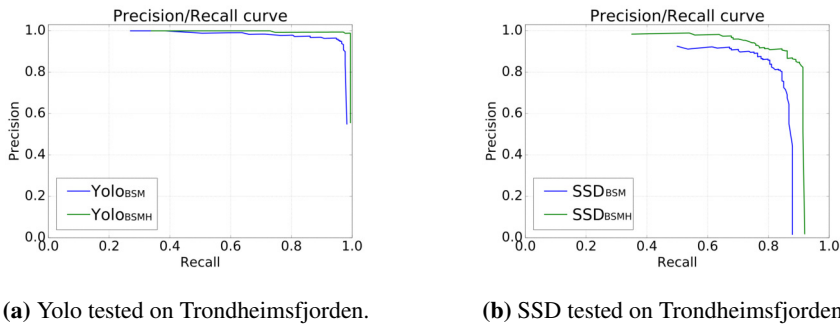


**Figure 6.6:** Yolo<sub>BSM</sub> misclassifies a cloud as a boat, Yolo<sub>BSMH</sub> does not. Red bounding boxes are detections, green bounding boxes are ground truth.

### 6.3.3 Tested on Trondheimsfjorden

The precision/recall curve for Yolo<sub>BSM</sub>, Yolo<sub>BSMH</sub>, SSD<sub>BSM</sub>, and SSD<sub>BSMH</sub> on Trondheimsfjorden can be seen in figure 6.7. All the models perform very well on this test set, where Yolo<sub>BSMH</sub> almost gets a perfect score with an average precision of 0.908, meaning it is very close to detect all the objects while having practically no misclassifications. In table 6.4 it is shown that Yolo<sub>BSMH</sub> detects 173 of 174 boats in the Trondheimsfjorden test set with a confidence threshold of 0.25.

Both the models trained on the building class, perform better than the model only trained on the boat class. As for the tests on BoatsClose and BoatsFar the differences between SSD<sub>BSM</sub> and SSD<sub>BSMH</sub> are more evident than the differences between Yolo<sub>BSM</sub> and Yolo<sub>BSMH</sub>. The performance difference between SSD<sub>BSM</sub> and SSD<sub>BSMH</sub> is also more significant than between Yolo<sub>BSM</sub> and Yolo<sub>BSMH</sub>, as can be seen in figure 6.7



**Figure 6.7:** Precision/recall curves for Yolo<sub>BSM</sub>, Yolo<sub>BSMH</sub>, SSD<sub>BSM</sub> and SSD<sub>BSMH</sub> on Trondheimsfjorden

### SSD<sub>BSM</sub> and SSD<sub>BSMH</sub> on Trondheimsfjorden

SSD<sub>BSM</sub> has some of the same behaviour on Trondheimsfjorden as it had on BoatsClose and BoatsFar. For instance, it sometimes classifies land as a boat, as shown in figure 6.8. This happens in approximately 5 percent of the test images. More examples of this behaviour can be seen in appendix 6.8.



**Figure 6.8:**  $SSD_{BSM}$  misclassifies land as boat.

There are examples where  $SSD_{BSM}$  detects boats correctly while  $SSD_{BSMH}$  don't and vice versa. For example images of this behaviour see Appendix C, chapter C.3.2 and C.3.3. In a few test images,  $SSD_{BSMH}$  wrongly classifies buildings as boats, while  $SSD_{BSM}$  does not. This is counter-intuitive since the idea behind training  $SSD_{BSMH}$  on buildings was to make it suppress these detections. An example of this is shown in figure 6.9.

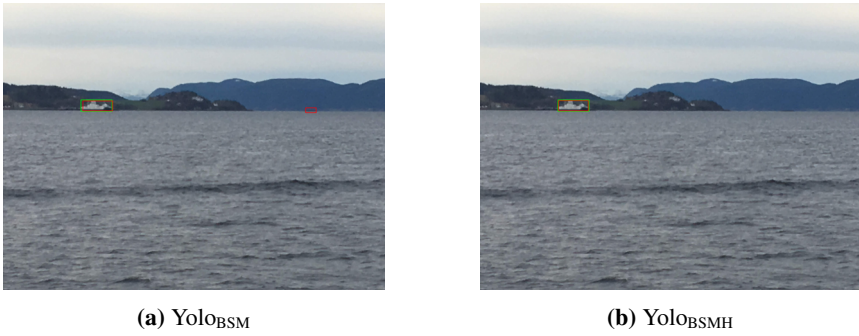


**Figure 6.9:**  $SSD_{BSMH}$  misclassifies two buildings as boats

In figure 6.9,  $SSD_{BSMH}$  does not detect the buildings as buildings, this will be discussed further in chapter 7.1.1.

### **Yolo<sub>BSM</sub> and Yolo<sub>BSMH</sub> tested on Trondheimsfjorden**

Yolo<sub>BSM</sub> and Yolo<sub>BSMH</sub> both have outstanding results on Trondheimsfjorden, but Yolo<sub>BSM</sub> has a few misclassifications that Yolo<sub>BSMH</sub> does not have. In figure 6.10 Yolo<sub>BSM</sub> detects land as a boat, and in figure 6.11 Yolo<sub>BSMH</sub> detects a boat that Yolo<sub>BSM</sub> does not. There are not many examples where their performance differs much, but there are a few which makes Yolo<sub>BSMH</sub> a little more robust than Yolo<sub>BSM</sub>.

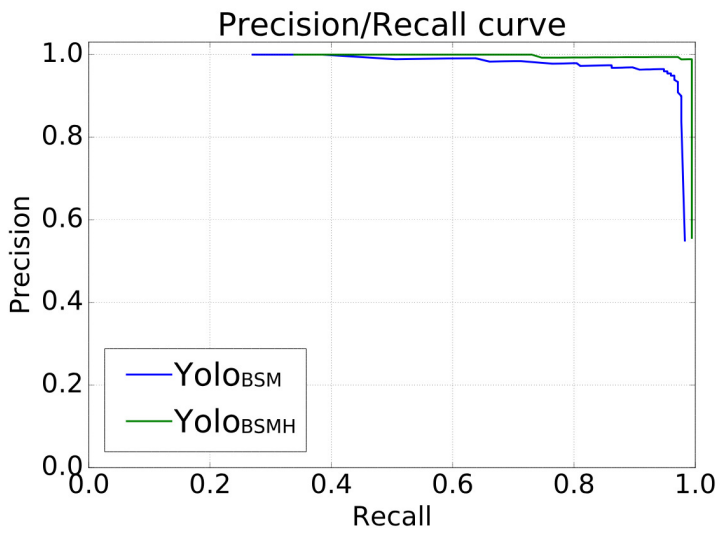


**Figure 6.10:** Yolo<sub>BSM</sub> misclassifies land as boat



**Figure 6.11:** Yolo<sub>BSM</sub> does not detect rightmost boat, Yolo<sub>BSMH</sub> does

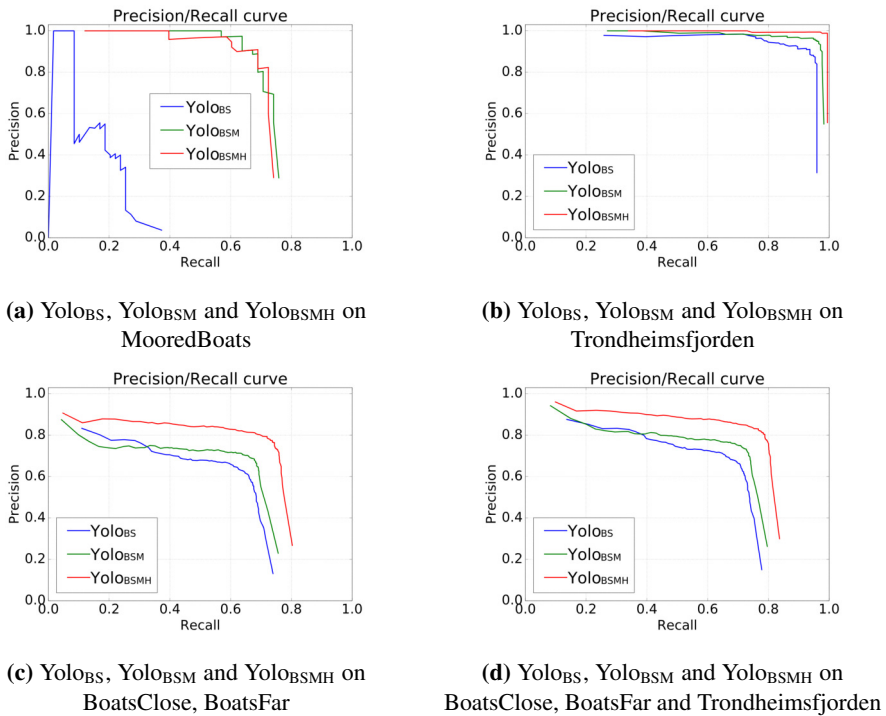
Summed up, the effect of training on buildings seems to have a positive influence on the results. Yolo<sub>BSMH</sub> outperforms Yolo<sub>BSM</sub> on all test datasets, and thus, this could indicate that training on a building class is beneficial for a boat detector. However, training on the building dataset also provides negative information for boat detection. By training on the building dataset, where there are no boats, the detector can also extract information about where there are no boats. Yolo<sub>BSMH</sub> is trained on the same boat objects as Yolo<sub>BSM</sub>, but Yolo<sub>BSMH</sub> could extract contextual information about the images in the building dataset, making it more able to identify objects that are not boats. This also conforms with the plot in figure 6.13, where the Yolo<sub>BSMH</sub>'s precision is higher than Yolo<sub>BSM</sub>'s, meaning Yolo<sub>BSMH</sub> has fewer misclassifications.



**Figure 6.13:** Precision/recall curves for Yolo<sub>BSM</sub> and Yolo<sub>BSMH</sub> on Trondheimsfjorden.

## 6.4 Case Study 2: Effect of Training on Moored Boats While Testing for Sailing Boats

Since  $\text{Yolo}_{\text{BSMH}}$ 's results on Trondheimsfjorden are nearly perfect, as shown in figure 6.14b, it is necessary to investigate if the model have been fitted to this type of data to a larger extent than it should. Therefore, a case study of how training on MooredBoats affect the results on detection of sailing boats (BoatsClose, BoatsFar, Trondheimsfjorden), was done.  $\text{Yolo}_{\text{BSM}}$  and  $\text{Yolo}_{\text{BSMH}}$  performs better on Trondheimsfjorden than  $\text{Yolo}_{\text{BS}}$ , without training on more data from Trondheimsfjorden. This could imply that the improved results is not connected to overtraining. This will be further discussed in chapter 7.1.



**Figure 6.14**

In figure 6.14 the precision/recall curves for  $\text{Yolo}_{\text{BS}}$ ,  $\text{Yolo}_{\text{BSM}}$  and  $\text{Yolo}_{\text{BSMH}}$  are shown.  $\text{Yolo}_{\text{BSM}}$  and  $\text{Yolo}_{\text{BSMH}}$  are better than  $\text{Yolo}_{\text{BS}}$  in all the test cases. While this is expected behaviour when testing on MooredBoats, the results are not obvious for the other test cases.

Two clear differences between  $\text{Yolo}_{\text{BS}}$  and  $\text{Yolo}_{\text{BSM}}$  were found while analyzing the results. The first one being that  $\text{Yolo}_{\text{BS}}$  tends to overestimate the size of large ships, as shown in figure 6.15. More examples of this behaviour can be seen in Appendix C, chapter C.4.1





**Figure 6.15:** Example of Yolo<sub>BS</sub> over estimating size of ship.

Yolo<sub>BS</sub> also has a penchant to detect the same boat twice, where Yolo<sub>BSM</sub> does not, as shown in figure 6.16. More examples of this behaviour are shown in Appendix C, chapter C.4.2.



**Figure 6.16:** Example of Yolo<sub>BS</sub> detecting same boat twice.

There seems to be a correlation between training on moored boats and improved performance when testing on sailing boats. As mentioned in case study 1, this could be caused by Yolo<sub>BSM</sub> and Yolo<sub>BSMH</sub> being trained on more data, even when this data is not the exact same as the test data. Yolo<sub>BS</sub> has seen less boats in training and could therefore be less robust than Yolo<sub>BSM</sub> and Yolo<sub>BSMH</sub>.

## 6.5 Case Study 3: Video and Temporal Data

In (Bøhn, 2018) the issue of detections in radar data over time is addressed. For a detector to work on temporal data, the object of interest must be detected with some frequency. If a boat is detected every other frame, it could probably be detected consistently using a tracking algorithm, as done in (Jiang and Singh, 2016). In this work a tracking algorithm has not been implemented. However, an experiment on how well boats are detected over time has been done.

### 6.5.1 Video from Trondheimsfjorden

Three videos of boats in Trondheimsfjorden have been captured, each approximately 30 seconds long. YOLO<sub>BS</sub> was tested on each of them. The three videos are merged and available on YouTube <sup>1</sup>. Frames from each of the videos are shown in figure 6.17.

---

<sup>1</sup><https://www.youtube.com/watch?v=kcirhao.PQc>



(a) Video of small boat (video<sub>SB</sub>)



(b) Video of big boat (video<sub>BB</sub>)



(c) Video of three boat (video<sub>3B</sub>)

**Figure 6.17:** Example frames from video<sub>SB</sub>, video<sub>BB</sub> and video<sub>3B</sub>. Detections shown as red bounding boxes.

In  $\text{video}_{\text{SB}}$  and  $\text{video}_{\text{BB}}$  the boat is detected correctly in every frame. Thus a tracking algorithm would not be needed to identify undetected boats in these videos. In both  $\text{video}_{\text{SB}}$  and  $\text{video}_{\text{BB}}$ , there is only one boat present. In  $\text{video}_{\text{3B}}$  there are three boats, where one of the boats passes behind another during the video.  $\text{Yolo}_{\text{BS}}$  detects the boats in all frames where the boats are separated, as shown in figure 6.18a. When the third boat is behind the second, and beginning to emerge behind it,  $\text{Yolo}_{\text{BS}}$  detects the two boats as one. This is shown in figure 6.18b. When the third boat is only slightly visible by human assessment,  $\text{Yolo}_{\text{BS}}$  only detects two boats as shown in figure 6.18c.



(a) Three separated boats



(b) Part of third boat clearly visible



(c) Third boat slightly visible

**Figure 6.18:** Example frames from video<sub>3b</sub> showing three different occurring scenarios in the video.

In video<sub>3B</sub> all parts of boats are detected as boats, though two boats are classified as one big boat instead of two smaller ones when one is in the shadow of the other. This is a hard problem to solve for an object detector alone, and will be discussed further in chapter 7.2.3.

## 6.5.2 Video from (Kamsvåg, 2018)

In (Kamsvåg, 2018) Faster R-CNN was implemented for boat detection. One of the prominent problems encountered in this work was the misclassification of buildings as boats. This was one of the main reasons to train SSD and Yolo on a building class, to see if this could help suppress these detections. Yolo<sub>BS</sub>, Yolo<sub>B<sub>SM</sub></sub> and Yolo<sub>B<sub>SMH</sub></sub> were run on a video used in (Kamsvåg, 2018), and the results can be found on YouTube <sup>2</sup>. Example frames from the video are shown in figure 6.19.

In this video the images are captured much closer to sea level than the ones used in training, which may have affected the results. Yolo<sub>BS</sub>, Yolo<sub>B<sub>SM</sub></sub> and Yolo<sub>B<sub>SMH</sub></sub> do all have some problems with detecting the boats in every frame, while Yolo<sub>BS</sub> has the poorest performance. Yolo<sub>B<sub>SMH</sub></sub> and Yolo<sub>B<sub>SM</sub></sub> have similar results on boat detection, but Yolo<sub>B<sub>SMH</sub></sub> has more misclassifications of buildings as boats than Yolo<sub>B<sub>SM</sub></sub>. Yolo<sub>B<sub>SMH</sub></sub> classifies some of the buildings as buildings, but this does not seem to help suppress the wrong identifications.

The purpose of training Yolo<sub>B<sub>SMH</sub></sub> on a building class was for it to be more robust to misclassifications of boats as buildings. In the results shown in chapter 6.2, this was not the case. Yolo<sub>B<sub>SM</sub></sub>, which is not trained on the building class, has fewer detections of buildings as boats than Yolo<sub>B<sub>SMH</sub></sub>. Yolo<sub>BS</sub>, Yolo<sub>B<sub>SM</sub></sub> and Yolo<sub>B<sub>SMH</sub></sub> all have lower accuracy on this video compared to the results on the rest of the test data. This is not entirely unexpected, since the video from (Kamsvåg, 2018) is different from the training data in several ways. First of all, the video from (Kamsvåg, 2018) is captured from a small boat close to sea level. This makes the angle between the center of the camera and the center of the boats vastly different compared to the ones in e.g. the Trondheimsfjorden dataset, where the images are taken from the top of a ferry. While the features of the boat does not change much due to the change of angle, the background is different, especially in coast-near environments. When the pictures are captured from a higher point above sea level the background of the boat will be more uniform, and in some cases consists of only water, while the background can be buildings or land in images taken closer to sea level as in figure 6.19. While this can be a factor to the weaker boat detections on the video from (Kamsvåg, 2018), it does not explain why Yolo<sub>B<sub>SMH</sub></sub> classifies more buildings as boats than Yolo<sub>B<sub>SM</sub></sub>. This being said, it should be noted that the detections of buildings as boats are not consistent over time, and could probably be filtered out in post processing.

---

<sup>2</sup><https://www.youtube.com/watch?v=A.qETwNuFYI>



(a) Yolo<sub>BS</sub>.



(b) Yolo<sub>BSM</sub>.



(c) Yolo<sub>BSMH</sub>.

**Figure 6.19:** Example frames from video from (Kamsvåg, 2018). Red bounding boxes are boat detections, blue boxes are building detections for Yolo<sub>BSMH</sub>.





# Discussion

## 7.1 Dataset diversity

As mentioned in chapter 6.3.3, Yolo performs almost perfectly on the Trondheimsfjorden dataset, which could be an indication of overtraining. The Trondheimsfjorden dataset consists of approximately 500 images taken in the Trondheimfjord during a ferry trip back and forth between Trondheim and Brekstad. The images were all captured the same day, under the same weather and lighting conditions. Also, due to a limited amount of boats at sea during the ferry trip, multiple images were captured of the same boats. Thus even though the images the detection models were tested on and trained on were separated, the images used in both testing and training had some common features. While this might give the impression that the results are biased, and not representative of how the algorithm would perform in a completely new environment, some arguments would suggest otherwise.

First of all, as shown in chapter 6.4,  $Yolo_{BSM}$  and  $Yolo_{BSMH}$  perform better than  $Yolo_{BS}$  on the Trondheimsfjorden test set. All these detection models are trained on datasets BoatsClose, BoatsFar and Trondheimsfjorden, while  $Yolo_{BSM}$  and  $Yolo_{BSMH}$  are also trained on other datasets that do not contain similar images as Trondheimsfjorden. This could imply that what improves the results of  $Yolo_{BSM}$  and  $Yolo_{BSMH}$  on Trondheimsfjorden, is not overtraining on the Trondheimsfjorden dataset, but rather the ability to recognize features in more diverse environments.

All the Yolo models has less accurate results on the BoatsClose and BoatsFar test set compared to Trondheimsfjorden. In BoatsClose on BoatsFar Yolo gets good results on images that contain separated boats, similar to the photos in the Trondheimsfjorden dataset. What makes the results less impressive are primarily due to images that include clusters of small boats, as shown in figure 7.1.



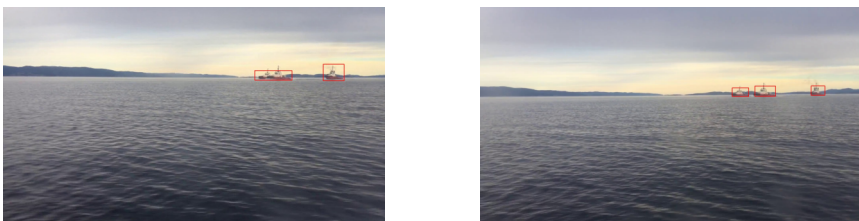
**Figure 7.1:** Yolo<sub>BSMH</sub> on cluttered small boats, red bounding boxes are detections, green bounding boxes are ground truth.

In the majority of the images, Yolo<sub>BSMH</sub> also gets excellent results in this test set, as shown in figure 7.2.



**Figure 7.2:** Yolo<sub>BSMH</sub> on separated boats in BoatsClose BoatsFar, red bounding boxes are detections, green bounding boxes are ground truth.

In the images in figure 7.1 it is not easy to correctly detect all the boats by human assessment either. When boats are separated, Yolo<sub>BSMH</sub> gets good results consistently. When vessels are more prominent, cluttering does not seem to be as problematic. This could partly be because big ships do not tend to stay very close together, and thus, the examples are fewer. An example of how Yolo<sub>BS</sub> handles three big tow boats close together is shown in 7.3.



(a) One tow boat partly covered by another.

(b) Three tow boats separated.

**Figure 7.3:** Yolo<sub>BS</sub> on big boat clutter, red bounding boxes are detections, green bounding boxes are ground truth.

In both figure 7.3a and 7.3b there are three towboats. In figure 7.3a one of the boats is partly covered by another, and the two boats are detected as one. This is not necessarily a problem, and as soon as the boats are separated, they are detected as three boats.

One cannot rule out that the results are unaffected by overtraining. To verify if the detection model is overtrained or not one would have to evaluate it in completely unseen

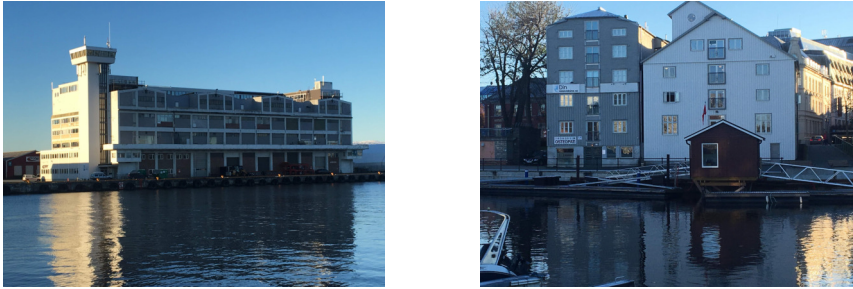
environments. At the same time, it does not serve any purpose to test the detection algorithm on a dataset that is nothing alike the Trondheimsfjorden dataset, since this is the environment the algorithm is meant to perform well in. The Trondheimsfjorden dataset contains many different types of boats and ships, it contains large camouflaged military vessels, small motor boats, and ferries, and is diverse relative to how different the boats and ships in the Trondheimfjord generally is.

The accuracy of the results Yolo<sub>BSMH</sub> has on the Trondheimsfjorden test set might also be because the detection problem of maritime vessels in the Trondheimfjord is not where Yolo struggles. It will depend on each case, but the results in this project indicate that Yolo can robustly detect sailing boats in the Trondheimfjord under good weather conditions. To verify this claim one would need an even more diverse test set, to ensure that the good results are not a consequence of overtraining. Still, there is no limit to how diverse a dataset could be and building a dataset that could verify this claim one hundred percent is not feasible with limited amount of time.

That being said, the weather conditions do not vary much, neither in the test or training dataset. Ideally, a dataset with a diverse set of ships and boats under different weather conditions such as dark lighting conditions, fog and snow should be made. The Cloud Detection Framework can be easily used to address the different scenarios, but the compilation of a dataset for these conditions should be done in future research on this topic.

### 7.1.1 Non-detected buildings in Trondheimsfjorden

In the Trondheimsfjorden dataset, there are buildings in the background which  $\text{Yolo}_{\text{BSMH}}$  does not detect, even though it is trained on a building class. The building dataset contains images taken in Trondheim close to Nidelva and a canal, and the buildings are in the majority of the images close up, as shown in figure 7.4



**Figure 7.4:** Example images from the buildings dataset.

In Trondheimsfjorden, there are examples where  $\text{Yolo}_{\text{BSMH}}$  fails to detect buildings, as shown in figure 7.5.



**Figure 7.5:**  $\text{Yolo}_{\text{BSMH}}$  fails to detect buildings in the background.

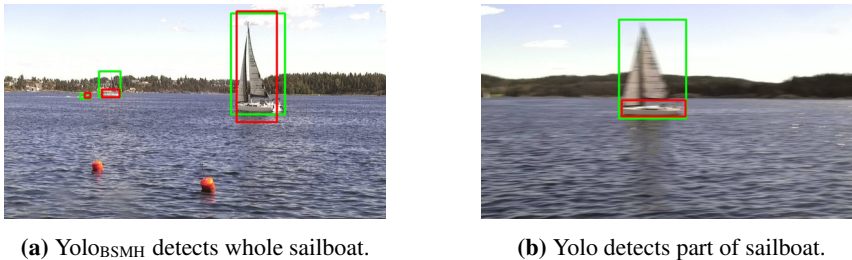
The buildings in figure 7.4 and 7.5 are in different scales. The results in this project indicate that training on images like the ones in figure 7.4 do not translate well to detecting buildings like the ones in figure 7.5. Still, the idea behind training a detection model on a building dataset was not that it should detect all the buildings in test datasets, but rather learn to not classify buildings as boats.

## 7.2 Labeling

During labeling of the dataset, the guidelines mentioned in chapter 4.1.2 was used. Three problems were encountered during the work with this project related to how the boats were labeled

### 7.2.1 Labeling of sail boats

In figure 7.6 there are examples where sailboats are detected differently. In figure 7.6a the whole sailboat is detected, with mast, in figure 7.6b the mast is not detected as a part of the boat. As can be seen in both figure 7.6a and 7.6b sailboats are labeled with mast. Thus, the correct detection set by the labeling standards would include the mast, however, to detect the mast may not be necessary for an autonomous vessel. The shape of the sailboat hull is comparable to the hulls of motor boats and also larger ships. By labeling sailboats with the mast, instead of only the hull of the boat, the detector might be less likely to be able to generalize the hull shape. Labeling only the sailboat hull makes the boats more alike, which may make the detector model more certain when classifying.



**Figure 7.6:** YoloBSMH detects two sailboats differently.

### 7.2.2 Using several boat classes

Another option is to train on different boat classes, having one class for motor vessels, sailboats, cargo ships, kayaks and so on. The features of a kayak will be different from a military frigate and trying to classify all these as the same object might confuse the detector. Yet, this would require datasets for each boat class, which would require much more data than what was possible to collect in this project.

### 7.2.3 Detecting boat parts

The detection models seem to struggle when boats are partly covered by other boats or objects, and will either only detect the foremost boat, or detect the two boats as one large object, as shown in figure 6.18. To be able to correctly classify a boat that is partly covered by another boat, the detection algorithm has to recognize parts of boats as boats. If large parts of a boat are covered, a detector trained on non-covered boats will not robustly detect it. A possible solution could be to train an object detector on parts of boats and post-process the detections into full boat detections. This does not seem like an easy system to implement and would require a massive amount of data.

### 7.3 Clustered boats

Both Yolo and SSD have good results when there is a limited amount of boats in the image, and they are separated. When small boats are clustered together, like in figure 7.1 both detection algorithms have trouble detecting them. This is not surprising as it is hard to make out the boats individually, also by human assessment.

For Yolo, there are two main ways of approaching this problem. Either the grid can be resized and that way Yolo can detect smaller objects. This because there will be more cells in the grid, thereby creating more object proposals. The other solution, which might work better in this case, is to identify where there might be clusters and run Yolo on this sub-region of the image. This would require some form of cluster detection, which has been done in (Van Etten, 2016), though on detection of boats in satellite images.

## Conclusion and Future Work

This project consists of two main contributions. First, the Cloud Detection Framework, which simplifies the process of training and testing of deep learning-based detection algorithms on different data. A considerable amount of the work done in this project has been spent implementing the Cloud Detection Framework. To verify whether an object detector is sufficiently robust to be used on an autonomous ferry, tests in different environments, weather conditions and locations needs to be done. Also, the data used in training of the object detector greatly affects its performance. Hence, a system which makes the process of training and testing more efficient may be of great value in the continued research on this topic.

The other main contribution was the collection and labeling of a relatively large dataset, and training and testing on this data. The research done in this project gives an indication of where continued research should be focused, and sets a benchmark for new implementations. That way training on new data, and implementations of new models can easily be compared to previous models, and lets the user know if the new model was a step in the right direction or not. The results from this project shows that both Yolo and SSD has potential to detect boats robustly. There are some situations where the models does not perform ideally, but these cases are in most situations connected to the training data. The detection models respond well to training, and with a sufficiently diverse training dataset they have potential to contribute to an improved understanding of the surroundings of an autonomous ferry. However, this is something that needs to be investigated further. The limitations of the object detector should be discovered further, to better understand its performance in different scenarios. How well the results are in foggy weather conditions, at night or how close boats has to be before they are detected should be investigated further.

As mentioned in chapter 3.7, there has not been done much research on the limitations when it comes to deep learning-based object detectors in maritime environments. Neither has there been done extensive research on how different training data affects the results in different maritime environments. In this project a starting point for this has been done. While that datasets in this report does not include datasets in different weather and lighting conditions, it sets up a system which makes this easy to test in the future. It also provides trained models with performance statistics, which can serve as a benchmark for new implementations.

## **8.1 Future Work**

### **8.1.1 Test on Video and Real-Time Performance**

An important aspect that has not been tested in this paper is the processing rate of the algorithms on live video. While both Yolo and SSD states processing times in their papers, this should be verified on local hardware. However, since both papers report approximately the same processing rate (between 20-150 FPS depending on version of SSD and Yolo), it is more trustworthy than if they had reported differently.

### **8.1.2 More Data**

The most important matter in the future work on this project is to gather more data and conduct tests that continues to analyze Yolo, SSD and other detections algorithms potential. The only way to find the limitations of these models are to test and train them on new data to find what makes them better, and what makes them worse. The data should be gathered in relevant environments and be as diverse as possible, containing different weather, boats and locations.

### **8.1.3 Tracking**

There has not been implemented a tracking algorithm in this project. A tracking algorithm could be of used in post processing of the detections and could help create a more robust understanding of the situation. A tracking algorithm could counter the effect of lost detections for few frames, as discussed in 6.5.2.



---

# Appendix **A**

---

## Setting up Google cloud

There are several cloud services that provides GPU processing, and cloud computing, some of the most popular are Amazon AWS, Microsoft Azure and Google Cloud. These cloud services makes it possible to run code on customisable hardware, and they provide libraries and documentation that makes it easy to set up. In this project Google Cloud was chosen because it gives you 300 USD worth of credit when you open an account, which makes it possible to get to know the framework before you have to pay. That being said, the code being used in this project should be able to run on cloud services by other providers as well. This appendix explains how you set up an account on Google cloud, how you create an instance with the correct hardware, and how one can access a Google Cloud instance from the command line on your own computer.

### **A.1 Creating an account**

Go to `cloud.google.com` and sign in using a Google account. Press "try free", and fill out the form. This will set up a project called "My first project"

### **A.2 Setting up an instance**

After setting up your account, go to the navigation menu in the top left corner, and go to compute engine. Go to VM instances and press create instance. Your screen should look like figure A.1.

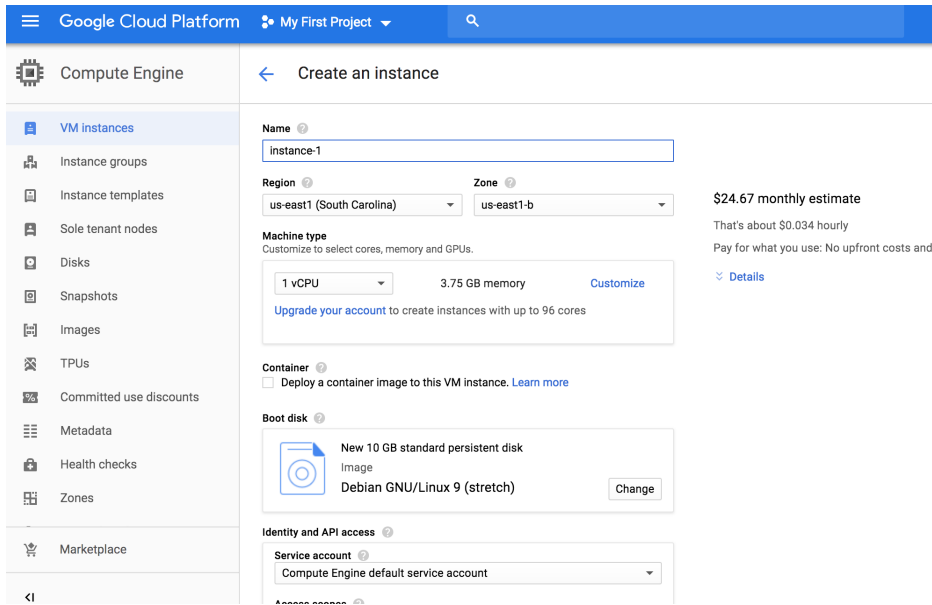


Figure A.1: Create instance

A configuration that has been tested to work is shown in figure A.2. Here the "Machine type" has been configured with one Nvidia Tesla K80 GPU. Boot disk has been set to Ubuntu 16.04 LTS, and http traffic is allowed. In this example the region is set to "europe-west1(Belgium)", and the zone to "europe-west1-b". The region and zone will affect the hourly price, and some zones and regions does not provide GPUs. There might be different combinations that serves other projects better than the combination chosen in this project. Also the size of the hard drive has been changed from its default value of 10 GB to 100 GB. Which might be necessary when uploading data to the instance.

Press "create instance" and the site will be directed to figure A.3. The green symbol indicates that the instance is running.

**Name** [?](#)  
instance-1

**Region** [?](#) **Zone** [?](#)  
europe-west1 (Belgium) europe-west1-b

**Machine type**  
Customize to select cores, memory and GPUs.

**Cores** Basic view

1 vCPU 1 - 8

**Memory**

3,75 GB 1 - 6.5

Extend memory [?](#)

**CPU platform** [?](#)  
Automatic

**GPUs**  
The number of GPU dies is linked to the number of CPU cores and memory selected for this instance. For this machine type, you can select no fewer than 1 GPU die. [Learn more](#)


**Number of GPUs** **GPU type**  
1 NVIDIA Tesla K80

**i** Machines with GPUs can't migrate on host maintenance

[Choosing a machine type](#) <sup>L</sup>  
[Upgrade your account](#) to create instances with up to 96 cores

**Container** [?](#)  
 Deploy a container image to this VM instance. [Learn more](#)

**Boot disk** [?](#)

 New 100 GB standard persistent disk  
Image  
Ubuntu 16.04 LTS Change

**Identity and API access** [?](#)

**Service account** [?](#)  
Compute Engine default service account

**Access scopes** [?](#)

- Allow default access
- Allow full access to all Cloud APIs
- Set access for each API

**Firewall** [?](#)  
Add tags and firewall rules to allow specific network traffic from the Internet

- Allow HTTP traffic
- Allow HTTPS traffic

[Management, security, disks, networking, sole tenancy](#)

The following options have been customized:  
On host maintenance

Figure A.2: Tested configuration

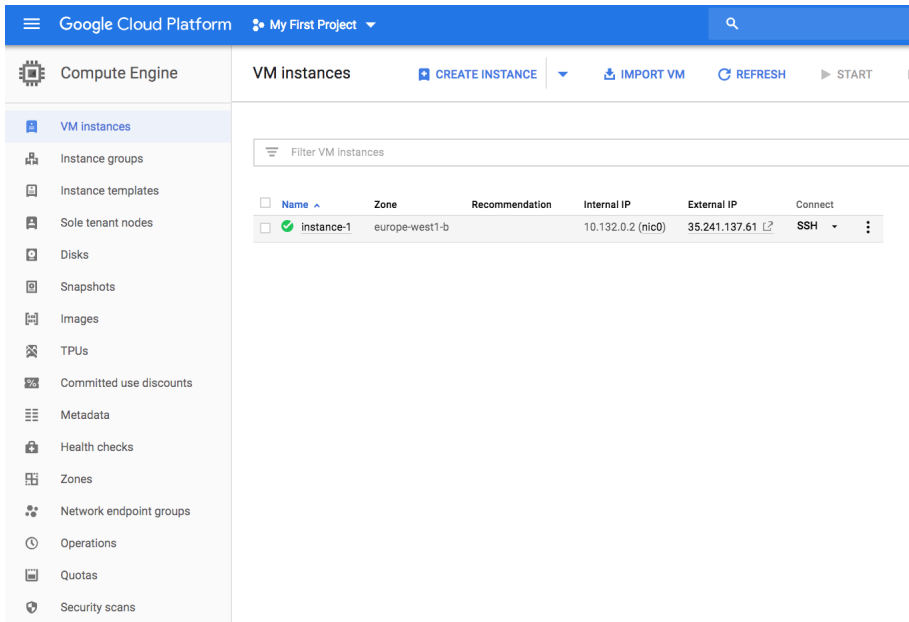


Figure A.3: Created instance

### A.3 Setting up and using gcloud command line tool on local computer

Install guides for all supported systems can be found on [cloud.google.com/sdk/docs/quickstarts](https://cloud.google.com/sdk/docs/quickstarts). This guide will take you through the installation process and the initialization process. The initialization process connects the gcloud command line tool to your google cloud account, and lets you access your instances easily when set up correctly. You will be prompted to log in to your Google account, choose project and default zone. Project can be chosen to be the on Gcloud set up when creating your account, and will be the only choice if you did not create new projects. The default zone can be chosen based on your own preference, in this project "europe-west1-b" was used.

### A.4 Accessing an instance from local computer using Gcloud command line tool

When the gcloud command line tool is installed and set up correctly, it can be used to access your instance. The way it has been accessed in this project is through SSH, and with SCP to transfer files.

#### A.4.1 SSH in to instance

To SSH into an instance the following command can be used. `gcloud compute ssh <name>` and `gcloud compute ssh <zone>` should be the name and zone of your created instance, as shown in figure A.3.

```
gcloud compute ssh <name> --zone <zone>
```

```
gcloud compute ssh instance-1 --zone europe-west1-b
```

### A.4.2 SCP file to instance

To transfer files to an instance the following command can be used.

```
gcloud compute scp <path/to/local/file> \<\  
<name>:/path/to/remote/file --zone <zone>
```

```
gcloud compute scp /home/data/image.jpg \<\  
instance-1:/home/images --zone europe-west1-b
```

This command will transfer the file image.jpg to the images repository on the cloud instance.

### A.4.3 SCP repository to instance

To transfer repositories and their content the following command can be used

```
gcloud compute scp --recurse </path/to/local/repo> \<\  
<name>:</path/to/remote/repo --zone <zone>
```

```
gcloud compute scp --recurse /home/data instance-1:/home/ \<\  
--zone europe-west1-b
```

This command will transfer the "data" repository and its content to the home directory of the instance.



# Appendix **B**

## Using the Cloud Detection Framework

After accessing an instance with SSH, the instance can be set up using the Github repository `cloud-detection-framework`.

```
git clone https://github.com/simenvg/cloud\_detection\_framework
```

When the github repository has been cloned, run `setup.sh` with the following command from your root directory.

```
./cloud\_detection\_framework/setup.sh
```

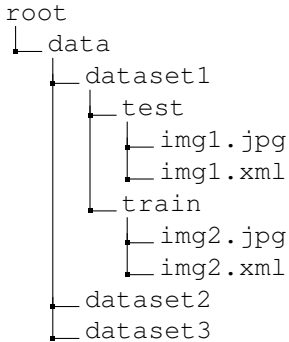
This will install CUDA, cudnn, tensorflow and other necessary libraries correctly, build darknet and set up an environment for training and testing both YOLOv3 and SSD. The setup file will run for approximately 10-15 minutes. There exists other releases of the libraries installed, that might be advantageous for other projects. The versions chosen here, are chosen because they are compatible with the hardware used, and work for both the SSD and YOLO implementation. There is no guarantee that anything will work, even after the smallest changes to this file or the hardware of the instance, as I have painfully

### **B.1 Training**

Both YOLO and SSD uses pretrained weights, this framework makes it possible to further train the networks on a custom dataset. The data can be uploaded to the cloud instance using SCP.

### B.1.1 Adding datasets and directory structure setup

In the cloud instance the following directory structure should be used for datasets



When adding a new dataset, it should be labelled using the VOC format annotation. This can be done using `labelImg`, from <https://github.com/tzutalin/labelImg>. This will produce an xml file for each jpg file labelled, with the same name. Put all the xml and jpg files in the same folder, and run the file `split_dataset.py` on the folder by running the following command.

```
python split_dataset.py /path/to/dataset
```

### B.1.2 Training YOLO

To train Yolo, run the file `train.py` in `cloud-detection-framework`. To run this file you need to provide the path to `darknet`, and the path to your data directory.

```
python train.py /home/user/darknet /home/user/data
```

This file will ask the user which datasets in the data directory Yolo should be trained on, and begin training.

### B.1.3 Training SSD

To train SSD two files need to be run. The first one being `convert_tfrecords.py`, and the second being



# Appendix C

## Example images

This appendix contains example images referenced in the thesis. The sub chapters are divided into different behaviours observed when testing the detection algorithms, and are connected to the case studies presented in chapter 6.

## C.1 $SSD_{BSM}$ and $SSD_{BSMH}$ on BoatsClose and BoatsFar



(a)  $SSD_{BSM}$



(b)  $SSD_{BSMH}$



(c)  $SSD_{BSM}$



(d)  $SSD_{BSMH}$



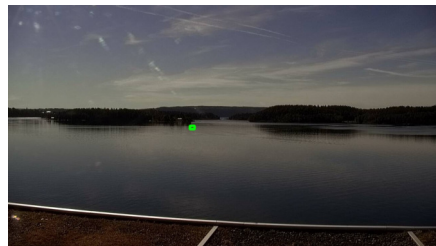
(e)  $SSD_{BSM}$



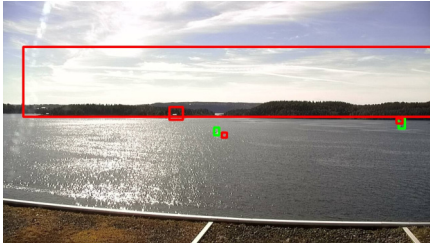
(f)  $SSD_{BSMH}$



(g)  $SSD_{BSM}$



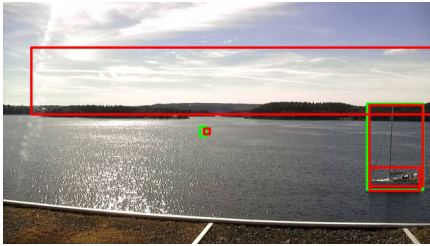
(h)  $SSD_{BSMH}$



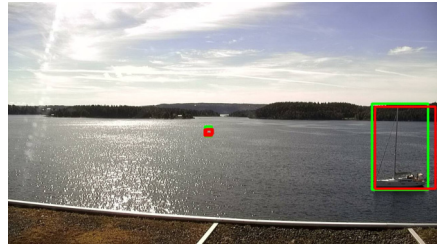
(a)  $SSD_{BSM}$



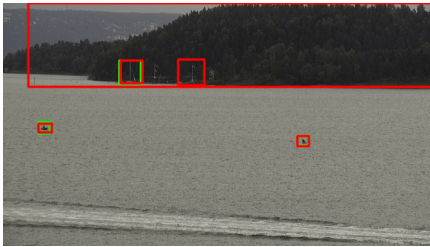
(b)  $SSD_{BSMH}$



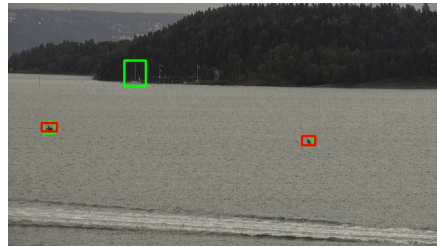
(c)  $SSD_{BSM}$



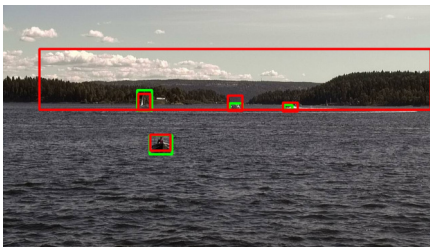
(d)  $SSD_{BSMH}$



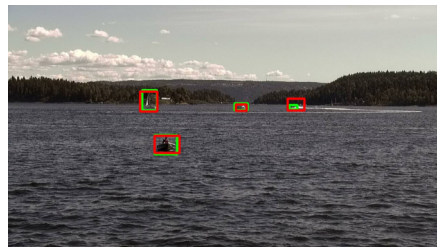
(e)  $SSD_{BSM}$



(f)  $SSD_{BSMH}$



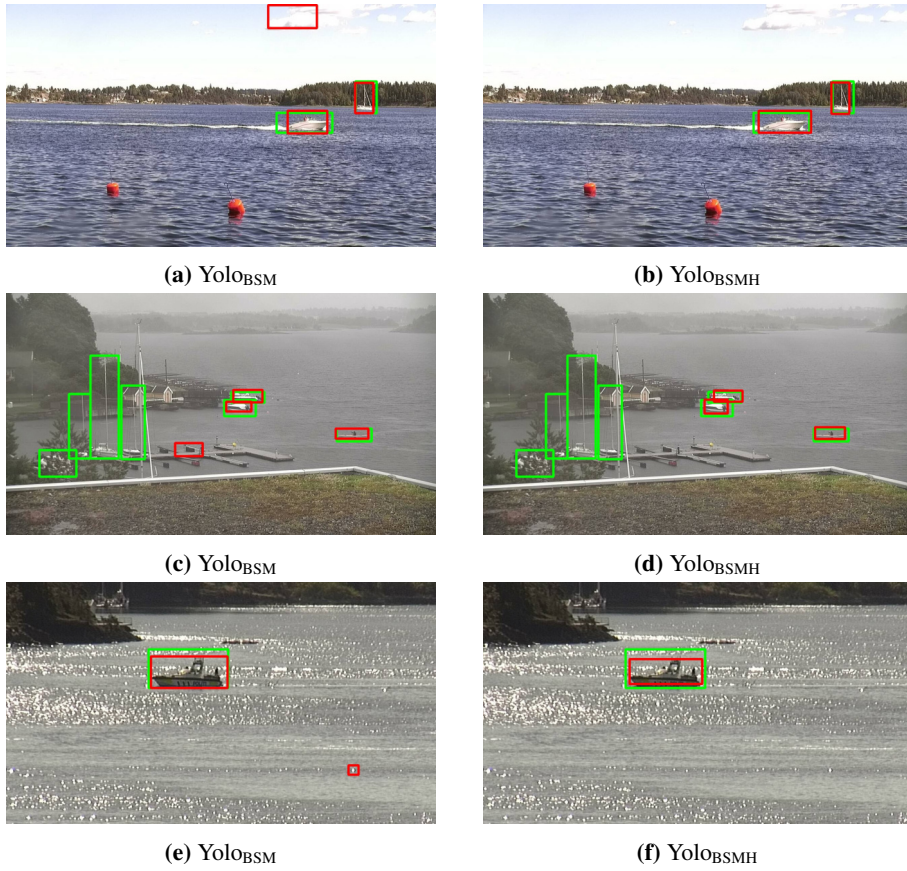
(g)  $SSD_{BSM}$



(h)  $SSD_{BSMH}$

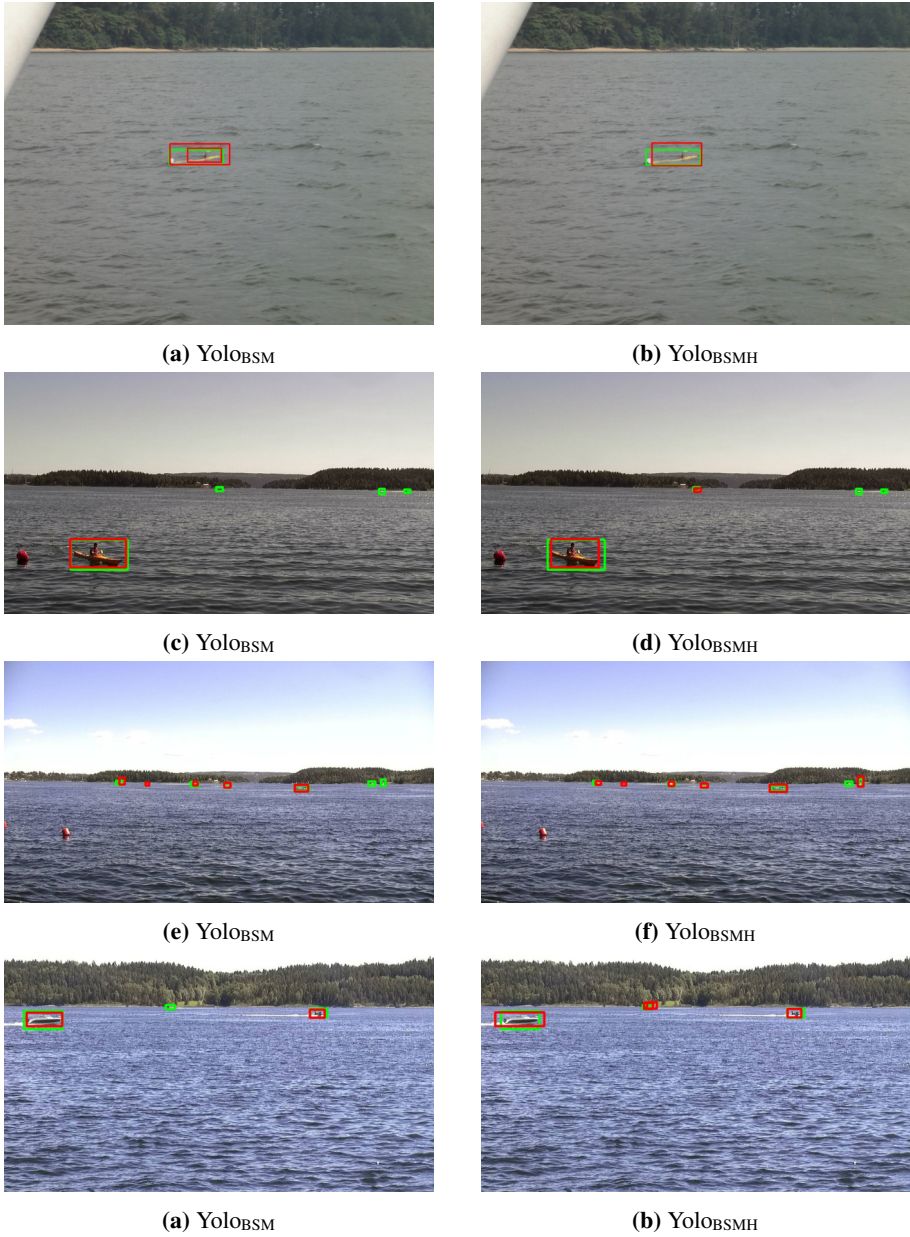
## C.2 Yolo<sub>BSM</sub> and Yolo<sub>BSMH</sub> on BoatsClose and BoatsFar

### C.2.1 Yolo<sub>BSM</sub> specific misclassifications on BoatsClose and BoatsFar



**Figure C.3:** Yolo<sub>BSM</sub> specific misclassifications on BoatsClose and BoatsFar

### C.2.2 Yolo<sub>BSMH</sub> better than Yolo<sub>BSM</sub> on BoatsClose and BoatsFar



**Figure C.5:** Example images where Yolo<sub>BSMH</sub> performs better than Yolo<sub>BSM</sub> on BoatsClose and BoatsFar.

### C.2.3 $Yolo_{BSM}$ better than $Yolo_{BSMH}$ on BoatsClose and BoatsFar



**Figure C.7:** Example images where  $Yolo_{BSM}$  performs better than  $Yolo_{BSMH}$  on BoatsClose and BoatsFar

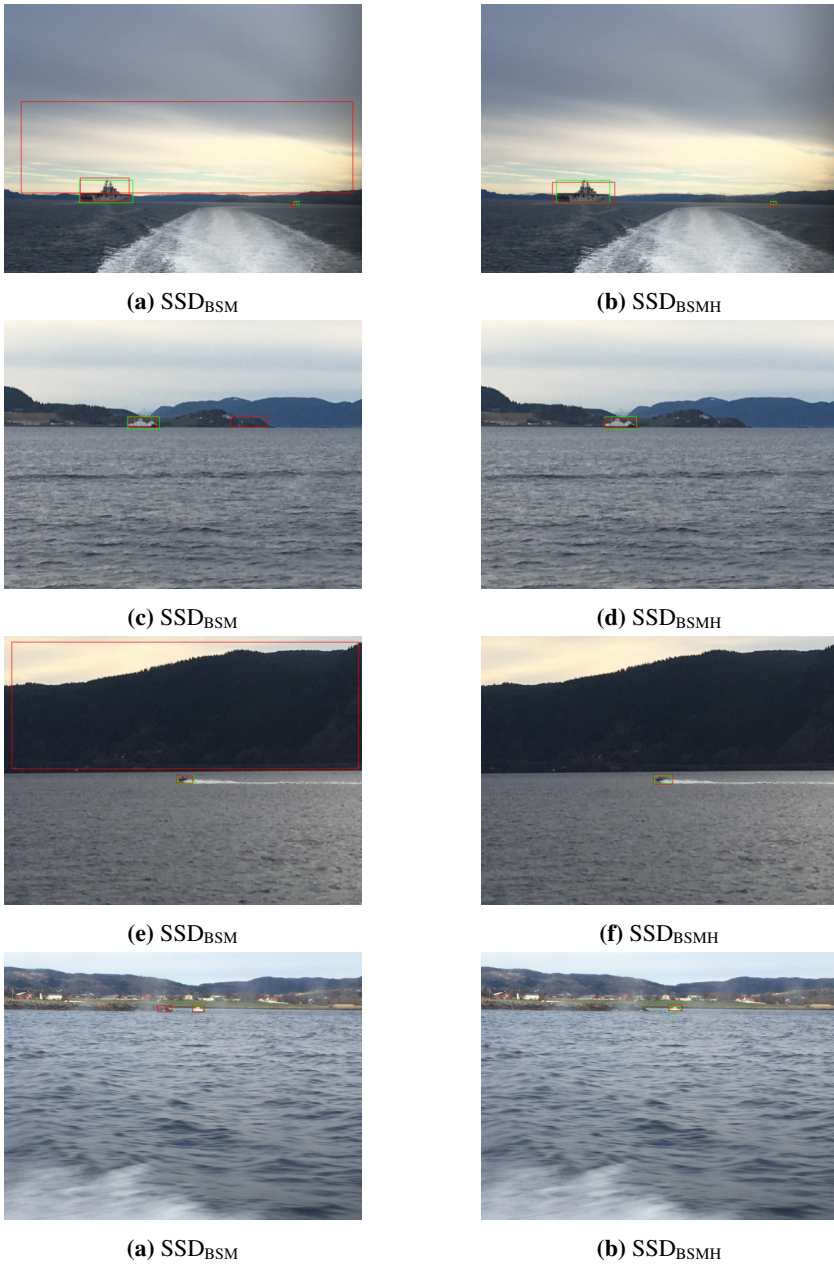
### C.2.4 Yolo<sub>BSM</sub> and Yolo<sub>BSMH</sub> same misclassifications on BoatsClose and BoatsFar



**Figure C.9:** Example images where Yolo<sub>BSM</sub> and Yolo<sub>BSMH</sub> misclassifies the same object.

### C.3 $SSD_{BSM}$ and $SSD_{BSMH}$ on Trondheimsfjorden

#### C.3.1 $SSD_{BSM}$ specific misclassifications on Trondheimsfjorden



**Figure C.11:** Example images with  $SSD_{BSM}$  specific misclassifications



C.3.2  $SSD_{BSMH}$  better than  $SSD_{BSM}$  on Trondheimsfjorden



(a)  $SSD_{BSM}$



(b)  $SSD_{BSMH}$



(c)  $SSD_{BSM}$



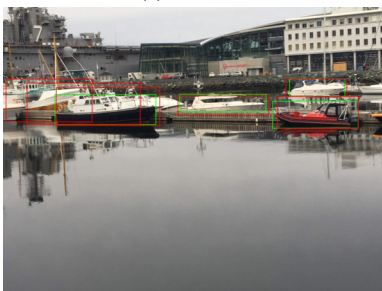
(d)  $SSD_{BSMH}$



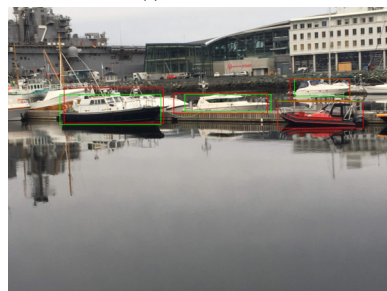
(e)  $SSD_{BSM}$



(f)  $SSD_{BSMH}$



(a)  $SSD_{BSM}$



(b)  $SSD_{BSMH}$

**Figure C.13:** Example images where  $SSD_{BSMH}$  performs better than  $SSD_{BSM}$

### C.3.3 $SSD_{BSM}$ better than $SSD_{BSMH}$ on Trondheimsfjorden



**Figure C.14:** Example images where  $SSD_{BSM}$  performs better than  $SSD_{BSMH}$ .

## C.4 Yolo<sub>BS</sub> and Yolo<sub>BSM</sub> differences

### C.4.1 Yolo<sub>BS</sub> overestimating ship size



(a) Yolo<sub>BS</sub>



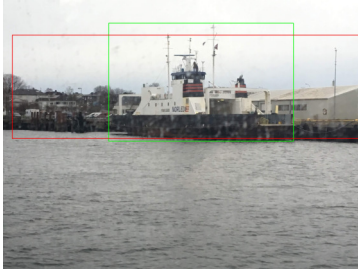
(b) Yolo<sub>BSM</sub>



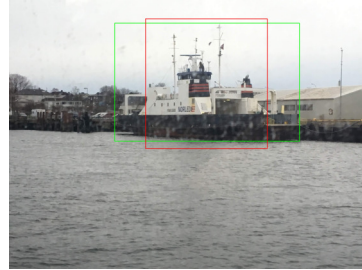
(c) Yolo<sub>BS</sub>



(d) Yolo<sub>BSM</sub>



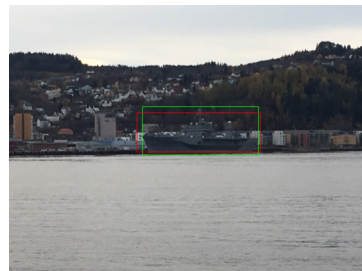
(e) Yolo<sub>BS</sub>



(f) Yolo<sub>BSM</sub>



(g) Yolo<sub>BS</sub>



(h) Yolo<sub>BSM</sub>

**Figure C.15:** Example images where Yolo<sub>BS</sub> overestimates size of ship, while Yolo<sub>BSM</sub> does not.

---

## C.4.2 Yolo<sub>BS</sub> detecting same boat multiple times



(a) Yolo<sub>BS</sub>



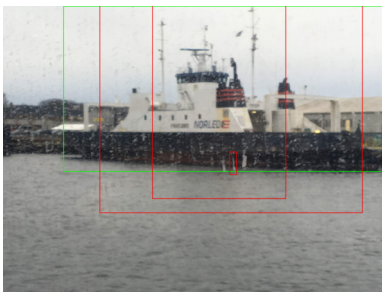
(b) Yolo<sub>BSM</sub>



(c) Yolo<sub>BS</sub>



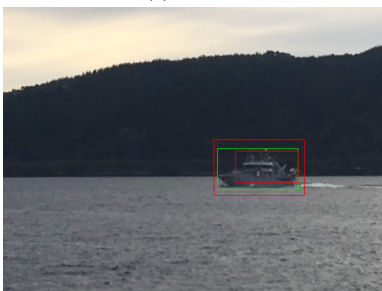
(d) Yolo<sub>BSM</sub>



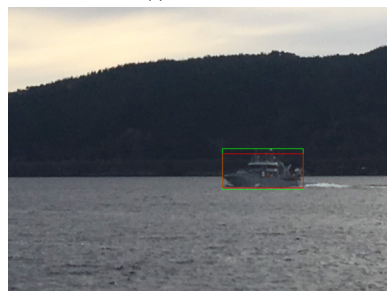
(e) Yolo<sub>BS</sub>



(f) Yolo<sub>BSM</sub>



(g) Yolo<sub>BS</sub>



(h) Yolo<sub>BSM</sub>

**Figure C.16:** Example images where Yolo<sub>BS</sub> detects boat multiple times, while Yolo<sub>BSM</sub> does not.

# Bibliography

- D. Bloisi, L. Iocchi, M. Fiorini, and G. Graziano. Automatic maritime surveillance with visual target detection. *International Defense and Homeland Security Simulation Workshop, DHSS 2011, Held at the International Mediterranean and Latin American Modeling Multiconference, I3M 2011*, (c):141–145, 2011.
- E. Bøhn. Semantic Segmentation of Radar Data with Deep Learning. 2018.
- J. Dai, Y. Li, K. He, and J. Sun. R-FCN: Object Detection via Region-based Fully Convolutional Networks. may 2016.
- J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. 2009a.
- J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. Technical report, 2009b.
- T. Dettmers. Deep Learning in a Nutshell: History and Training. 2015.
- R. Dulski, S. Milewski, M. Kastek, P. Trzaskawka, M. Szustakowski, W. Ciurapinski, and M. Zyczkowski. Detection of small surface vessels in near, medium, and far infrared spectral bands. volume 8185, page 81850U. International Society for Optics and Photonics, oct 2011. doi: 10.1117/12.898272.
- D. Erhan, C. Szegedy, A. Toshev, and D. Anguelov. Scalable Object Detection using Deep Neural Networks. Technical report, 2014.
- H. Eum, J. Bae, C. Yoon, and E. Kim. Ship Detection Using Edge-Based Segmentation and Histogram of Oriented Gradient with Ship Size Ratio. 15(4):251–259, 2015.
- M. Everingham and J. Winn. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Development Kit. 2007.
- M. Everingham and J. Winn. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Development Kit. 2012.
- M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (VOC) challenge. *International Journal of Computer Vision*, 88 (2):303–338, 2010. ISSN 09205691. doi: 10.1007/s11263-009-0275-4.

- 
- A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the KITTI vision benchmark suite. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 3354–3361, 2012. ISSN 10636919. doi: 10.1109/CVPR.2012.6248074.
- R. Girshick. Fast R-CNN. 2015.
- R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. nov 2013.
- R. Goring. Feasibility of Neural Networks for Maritime Visual Detection on a Mobile Platform. *Dissertations and Theses*, apr 2017.
- K. He, X. Zhang, S. Ren, and J. Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. 2015.
- K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask R-CNN. Technical report, 2018.
- L. Jiang and S. S. Singh. Tracking multiple moving objects in images using Markov Chain Monte Carlo. mar 2016.
- V. Kamsvåg. Fusion between camera and lidar for autonomous surface vehicles. 2018.
- A. Karpathy and L. Fei-Fei. Deep Visual-Semantic Alignments for Generating Image Descriptions. 2016.
- Kongsberg. Autonomous ship project, key facts about YARA Birkeland - Kongsberg Maritime, 2017. URL <https://www.km.kongsberg.com/ks/web/nokbg0240.nsf/AllWeb/4B8113B707A50A4FC125811D00407045?OpenDocument>.
- A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks, 2012.
- A. Kumar, S. Kushwaha, and R. Srivastava. Maritime Object Segmentation Using Dynamic Background Modeling and Shadow Suppression. 2015. doi: 10.1093/comjnl/bxv091.
- T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár. Microsoft COCO: Common Objects in Context. Technical report.
- T. Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: Common objects in context. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8693 LNCS(PART 5):740–755, may 2014. ISSN 16113349. doi: 10.1007/978-3-319-10602-1\_48.
- T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal Loss for Dense Object Detection. Technical report, 2018.
- W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. SSD: Single Shot MultiBox Detector. dec 2016. doi: 10.1007/978-3-319-46448-0\_2.

- 
- S. J. Pan and Q. Yang. A Survey on Transfer Learning. 2009. doi: 10.1109/TKDE.2009.191.
- N. Pires, J. Guinet, and E. Dusch. ASV : An innovative automatic system for maritime surveillance. *Navigation*, 58(232):1–9, 2010.
- D. K. Prasad, C. K. Prasath, D. Rajan, L. Rachmawati, E. Rajabally, and C. Quek. Challenges in video based object detection in maritime scenario using computer vision. Technical report, 2016a.
- D. K. Prasad, C. K. Prasath, D. Rajan, L. Rachmawati, E. Rajabally, and C. Quek. Challenges in video based object detection in maritime scenario using computer vision. 2016b.
- D. K. Prasad, D. Rajan, L. Rachmawati, E. Rajabally, and C. Quek. Video Processing from Electro-optical Sensors for Object Detection and Tracking in Maritime Environment: A Survey. nov 2016c.
- J. Redmon and A. Farhadi. YOLO9000: Better, Faster, Stronger. 2016.
- J. Redmon and A. Farhadi. YOLOv3: An Incremental Improvement. 2018.
- J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You Only Look Once: Unified, Real-Time Object Detection. 2016.
- S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. 2016.
- B. J. Rhodes, N. A. Bomberger, M. Seibert, and A. M. Waxman. SeeCoast: Automated port scene understanding facilitated by normalcy learning. *Proceedings - IEEE Military Communications Conference MILCOM*, 2007. ISSN 2155-7578. doi: 10.1109/MILCOM.2006.302306.
- P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks. dec 2013.
- D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the Game of Go with Deep Neural Networks and Tree Search. 2016.
- StanfordUniversity. Stanford University CS231n: Convolutional Neural Networks for Visual Recognition, 2018. URL <http://cs231n.stanford.edu/>.
- T. Stensvold. Verdens første førerløse passasjerferge kan gå over en kanal i Trondheim - Tu.no, 2016.
- Z. Sun, G. Bebis, and R. Miller. On-road vehicle detection: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(5):694–711. ISSN 01628828. doi: 10.1109/TPAMI.2006.104.
-

- 
- C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu. A Survey on Deep Transfer Learning. Technical report, 2018.
- E. J. Tangstad. Visual Detection of Maritime Vessels. 2017.
- T. H. Tran and T. L. Le. Vision based boat detection for maritime surveillance. *International Conference on Electronics, Information, and Communications, ICEIC 2016*, (January 2016), 2016. doi: 10.1109/ELINFOCOM.2016.7563033.
- J. R. R. Uijlings, K. E. A. Van De Sande, T. Gevers, and A. W. M. Smeulders. Selective Search for Object Recognition. 2012.
- A. Van Etten. Object Detection in Satellite Imagery, a Low Overhead Approach, Part I, 2016.
- A. Van Etten. You Only Look Twice: Rapid Multi-Scale Object Detection In Satellite Imagery. Technical report, 2018.
- a. Wedel and U. Franke. Monocular Video serves RADAR-based Emergency Braking. *2007 IEEE Intelligent Vehicles Symposium*, pages 93–98, 2007. ISSN 1931-0587. doi: 10.1109/IVS.2007.4290097.
- R. Zhang, P. Isola, and A. A. Efros. Colorful Image Colorization. 2016.
- M. H. Zwemer, R. G. J. Wijnhoven, and P. H. N. De With. Ship Detection in Harbour Surveillance based on Large-Scale Data and CNNs. 2018. doi: 10.5220/0006541501530160.



