NTNU
Norwegian University of
Science and Technology

Visual Detection of Maritime Vessels

Espen Johansen Tangstad

Master of Science in Cybernetics and Robotics
Submission date: June 2017
Supervisor: Edmund Førland Brekke, ITK
Co-supervisor: Andreas Lindahl Flåten, ITK
Håkon Hagen Helgesen, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Problem Formulation

Thesis Description

Obstacle avoidance is a requirement for autonomous ship operations. Therefore, other objects in the planned path of the ship need to be identified and tracked. Object detection with an optical sensor (camera) is one way to detect objects. However, this is challenging in maritime environments since objects at sea can be hard to distinguish from the waterline or hidden in waves. Furthermore, cameras have a limited range and objects far from the camera might be impossible to detect or cluttered by noise or other objects. In this project detection of marine vessels are the main priority. The algorithms developed in this project are not required to run in real-time, the focus is on detection performance and evaluation.

The following items should be considered:

1. Perform a literature review over camera detection theory, algorithms and techniques in maritime context.
2. Evaluate detection algorithms and propose a suitable solution for maritime object detection.
3. Implement a detection algorithm.
4. Test the detector's ability to detect objects.
5. Present the results and discuss limitations and challenges.

Start date: 09-01-2017
Due date: 06-06-2017
Thesis performed at: Department of Engineering Cybernetics, NTNU
Supervisor: Associate Professor Edmund Førland Brekke, NTNU
Co-supervisor: Andreas Lindahl Flåten, NTNU
Co-supervisor: Håkon Hagen Helgesen, NTNU

Abstract

Recent years have seen a large increase in the use of optical detection and tracking methods in autonomous cars. Both Google X's self-driving car and Tesla's semi-autonomous Model S rely heavily on the use of camera systems to help navigate complex environments on the roads. The camera captures enormous amounts of information from a scene, and the computer vision community is finding better and better approaches to extracting this information. The later years have seen an increase in the use of neural networks for the task of detecting and classifying objects in an image. These neural networks are known as convolutional neural networks (CNNs).

This thesis investigates how a tailored CNN can aid autonomous surface vehicles (ASVs) in detecting and classifying maritime traffic for collision avoidance. Several state of the art CNN models are presented and trained on data sets with relevance to the above-mentioned objective. Data collected from different sources are used for training these CNN models in pursuit to obtain a good performing detector. The main data sets are large, general purpose image sets of ships and boats. A smaller image set is also developed in this thesis. This custom data set is constructed from images taken along a predefined path at sea from a video camera. This includes images along docks and of ships in transit at sea. This data set is then split into training and testing images which are in close relation to each other. Through experiments, variations of the general purpose data sets are used to train both a 5 layer deep and a 16 layer deep CNN model to detect ships in an image.

The trained models are evaluated for their performance on a large test data set. This gives a comparable performance benchmark between the different models trained in the experiments on their ability to correctly detect ships in an image. The best performing model is then trained again, padded with the smaller, custom training data. Finally, all the models trained in the different experiments are evaluated on the custom test images. This evaluates the possibility of creating a more robust object detector for an autonomous ferry, travelling along a predefined path, trained on images from that path. Promising results were shown for both a general purpose ship detector and the detector developed for traversing a predefined path.

Keywords: Autonomous Surface Vehicle, Collision Avoidance, Vision Based Navigation, Convolutional Neural Network, Object Detection, Computer Vision, Image Processing

Sammendrag

De siste årene har vist en stor økning i bruk av optiske deteksjons og sporings metoder i autonome kjøretøy. Både Google X sin helautonome bil og Tesla's semi-autonome Model S anvender kamerasystemer som hjelpemiddel når de navigerer på veiene. Kameraet fanger enorme mengder informasjon om omgivelsene i bildet, og forskning innen datasyn gir stadig bedre metoder for å ekstrahere denne informasjonen. De siste årene har vist en økning i bruken av nevrale nettverk for deteksjon og klassifisering av objekter i bilder. Disse nevrale nettverkene er kjent som foldings nevrale nettverk (FNN).

Denne avhandlingen utforsker hvordan tilpassede FNN kan hjelpe autonome skip unngå kollisjoner ved å detektere og klassifisere den maritime trafikken i omgivelsene rundt skipet. Her presenteres flere av de nyeste FNN modellene. Disse trenes så på bildedata med relevans for det overnevnte formålet. Data samlet fra forskjellige kilder er brukt til trening i jakten på å oppnå en detektor med god ytelse. Hoveddataen kommer fra store bildesett av skip og båter. I tillegg til dette utvikles et mindre bildesett. Dette tilpassede bildesettet er bygd opp av bilder fra et videokamera som reiser langs Trondheimsfjorden og i Trondheim havn. Bildesettet er partisjonert i to data sett, ett for trening og ett for testing, hvor dataen i begge settene er i nær relasjon til hverandre. Gjennom eksperimenter trenes FNN modellene på forskjellige treningsdata fra de store bildesettene. Det trenes en 5 lags, samt en 16 lags dyp FNN modell til å detektere skip i bilder.

De trente modellene evalueres så for deres ytelse på store test bildesett. Dette gir en sammenlignbar ytelsesmåling mellom de forskjellige modellene trent i eksperimentene for deres evne til å detektere skip i bilder. Den beste modellen blir så trent på nytt, denne gangen med treningsbildene fra det mindre, tilpassede data settet. Til slutt blir alle modellene evaluert på det tilpassede testsettet. Denne evalueringen viser om den siste modellen som er trent på det tilpassede treningssettet har en forbedret ytelse over de generelle modellene. God ytelse for denne modellen gir muligheter å utvikle en mer lokal detektor for en autonom ferje som reiser langs en enkel rute, trent på bilder fra denne ruta. Lovende resultater presenteres, både for en generell skips-detektor og for en detektor som blir brukt langs en forhåndsbestemt rute.

Nøkkelord: Autonome Skip, Kollisjonsunngåelse, Kamerabasert Navigasjon, Foldings Nevrale Nettverk, Objekt Deteksjon, Datasyn, Bildebehandling

Preface

This thesis is carried out in the Department of Engineering Cybernetics, at the Norwegian University of Science and Technology, the spring of 2017. It is submitted as part of the requirements for the degree of MSc. at the Norwegian University of Science and Technology.

For valuable guidance throughout every stage of this thesis, i would like to thank Associate Professor Edmund Førland Brekke. In the same regard, this work would not have been possible without the help from my two co-supervisors, Andreas Lindahl Flåten and Håkon Hagen Helgesen.

Finally, I would like to thank all of my family for their support.

Trondheim, June 2017

Espen J. Tangstad

Table of Contents

List of Tables

List of Figures

List of Abbreviations

AIS	Automatic Identification System
AP	Average Precision
API	Application Programming Interface
ASV	Autonomous Surface Vehicle
CCD	Charge-Coupled Device
CNN	Convolutional Neural Network
COLREGS	Convention on the International Regulations for Preventing Collisions at Sea
ILSVRC	Imagenet Large-Scale Visual Recognition challenge
IoU	Intersection over Union
NMS	Non-Maximum Suppression
R-CNN	Region-based Convolutional Neural Network
RADAR	RAdio Detection And Ranging
ReLU	Rectified Linear Unit
RPN	Region Proposal Network
SGD	Stochastic Gradient Descent
UAV	Unmanned Aerial Vehicle
VGG	Visual Geometry Group

Part I

Background

Chapter 1

Introduction

Detection and tracking at sea have traditionally been based on data from Automatic Identification System (AIS) and RAdio Detection And Ranging (RADAR). In addition to AIS and RADAR navigation, it is desirable to investigate the possibility of using a computer vision system to detect and track maritime vessels. This will then serve as a complement to RADAR and AIS for maritime collision avoidance. AIS is only required for larger ships and tankers, which leaves smaller boats like sailboats undetected to the manoeuvring system. The position of these smaller vessels will be picked up by the RADAR, but with no information about the vessels structure or heading. A camera tracking system will provide this critical data, and help determine if the International Regulations for Preventing Collisions at Sea (COLREGS) is expected to be adhered to or not.

Camera-systems have been employed for many years, in various industrial tasks. As a sensor, the camera gathers a vast amount of information about a scene, and trying to exploit this data is no trivial task. It has been studied extensively for decades. We see improvements every year, as computers gain more processing power and the knowledge of the field expands. The exponential growth in computer power has allowed for complex approaches to extracting information from images, so complex in fact, that they were deemed to be only theoretical when first thought of. One of these approaches is the Neural Network.

The recent advancements in the field of Neural Networks has sparked the interest of a lot of people, and the field is expanded upon every year. The industrial applications have started to become more prominent in recent years. In Computer Vision, Neural Networks have started to improve upon classical means of extracting information from images. Employing Neural Networks for collision avoidance at sea, from a moving vessel has not been studied much. The need here is for a fast algorithm, with robust detection and classification, to be employed in real-time. Neural Networks are more or less exclusively trained and tested on static images, thus there is a need to evaluate the industrial application in moving camera systems. Artificial Neural Networks are known to be slow from input to output, solving rudimentary tasks in magnitudes of seconds, or even minutes. The recent

rise in Convolutional Neural Networks, sharing parameters between layers, have opened up for faster image processing, reducing the time to milliseconds performing the same task. The hypothesis is therefore that, Convolutional Neural Networks trained and evaluated on detecting and classifying objects in a static image will be suitable for detecting and classifying moving objects, in a moving scene, from a moving platform, in real-time.

The subject of extracting maritime vessels from images have been researched extensively, as can be seen in the survey, covering almost 500 papers in [7]. The general approach to extracting ship candidates from camera images is to segment the image using some segmentation algorithm and then doing some form of texture and/or shape analysis. Variations of this approach is proposed with satisfactory results given good weather conditions [12],[27]. Another means of achieving detection is using focused image correlation between two images in a video-stream from a UAV hovering over the sea [20]. In later years, classification and detection using Neural Networks have gotten increasing attention. Deep convolutional neural nets are starting to show a substantial improvement upon classical methods in both classification [21],[28],[34] and object detection [34],[16],[35],[15].

This thesis explores means to extract maritime vessels from images captured with a video camera at sea. The solution explored is a framework called Faster R-CNN [25]. Faster R-CNN is an all in one object detection and object classification framework. A image from the video-stream is extracted and fed through a convolution neural network (CNN) which proposes regions in the image. This CNN is called the region proposal network (RPN) and does binary classification on a set of regions in the image to determine if it belongs to a object or the background. The image, with its positive regions are then fed to another CNN called Fast R-CNN. Fast R-CNN classifies the regions, if they belong to background, or a set of classes defined by the user. In this thesis, these CNNs are trained and tested to provide insight into their applicability for detecting ships in pursuit of providing a robust detector of ships for autonomous surface vehicles (ASVs).

The contributions to this thesis are as follows:

- Literary study of object detection methods.
- Creating a testbench for training neural networks.
- Training several CNN models in pursuit of optimal performance.
- Creation of a custom data set used for training and testing.
- Defining and implementing a metric to determine the detection error distribution.
- Comprehensive discussion about pitfalls and lessons learned from using and training a CNN.

1.1 Outline

This thesis is organized in 9 chapters, as well as a appendix covering some implementation aspects. The chapters are numbered 1-9. Here is a short description of the contents of each chapter:

- Chapter 2 presents the camera as well as some fundamental image processing techniques. These include blurs and edge detectors.
- Chapter 3 presents the convolutional neural networks. A breakdown of each component substantiating a CNN is presented. This chapter also presents the training procedure used for optimizing a CNN called backpropagation. The last section explores some well established CNN models, which have shown high object classification score in the literature.
- Chapter 4 presents the framework used for implementation in this thesis, Faster R-CNN. The chapter presents some of the history leading up to the state of the art framework that is used in this thesis. Here, some of the main aspects of the framework is explored and the iterative training procedure is presented at the end.
- Chapter 5 presents the various data sets used to train the CNN models. It also presents a custom made data set designed specifically for evaluating robustness for a ASV.
- Chapter 6 covers the implementation of Faster R-CNN using MATLAB and the Caffe library. The chapter ends with some of the implementation aspects such as challenges and documentation.
- Chapter 7 describe the experiments conducted to evaluate the robustness of the different object detectors.
- Chapter 8 present the results from the different experiments with a discussion for each experiment evaluating the obtained results.
- Chapter 9 concludes the report. Here, the main findings are presented, as well as possible future work related to what has been presented.
- The appendix presents some key aspects for anyone looking to modify Faster R-CNN to train on their own data set.

Part II

Theory

Chapter 2

Image Processing

Camera systems are used extensively in the industry. It has a wide array of applications ranging from surveillance (CCTV), to image acquisition in both aerial, space and underwater scientific explorations. In recent years, camera-systems have been employed to aid humans and robots alike in detecting and classifying objects in the real world. This chapter presents a brief introduction to the camera and how it acquires color images. It also gives an introduction to some important image processing techniques. This chapter is targeted towards readers with little experience in image processing.

- Section 2.1 introduces the camera and some key aspects of camera operation.
- Section 2.2 presents image processing techniques such as Gaussian smoothing and edge detectors.

2.1 Image acquisition

The modern camera uses a square array of color filters consisting of red, blue and green. These filters are positioned over an array of photosensors. When the camera is exposed to light, the photosensors are excited, converting the photons into current. This current, in turn, using a ADC-converter may be read by a microcontroller as an intensity-value, usually 8-bit, ranging from 0-255. The most common structure for the color filter array is known as the bayer filter. This is shown in Figure 2.1. It consists of a repeated 2x2 pattern of red, blue and 2 times green. The complimentary green filter is added because the human eye is more sensitive to green light compared to red and blue [8].

2.2 Image processing

Image processing in the broadest term is the operation of applying some sort of filter to change image properties. This is done by treating the image as a 3-dimensional signal (in the case of RGB images) and applying filter convolutions to reduce the complexity of the

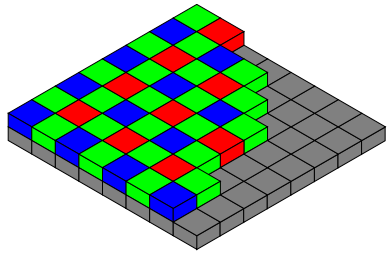


Image courtesy: en.wikipedia.org/wiki/Bayer_filter

Figure 2.1: The Bayer filter

image. An image from a video-stream may be treated like a four-dimensional input, where the last dimension is time. Image processing can be an important first step in machine-vision. This section aims to provide some insight into how filter convolutions change the properties of images by investigating two distinct image processing operations.

2.2.1 Smoothing

Gaussian smoothing is the operation of convolving a 2-D Gaussian operator with the image to reduce noise in the image. The continuous 2-D Gaussian operator is given by:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.1}$$

To discretize, the continuous Gaussian kernel is sampled from its maxima a number of samples in each direction, circularly to obtain a square $u \times v$ matrix. It is then normalized to obtain a discrete approximation of the Gaussian:

$$G_d = \frac{1}{289}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 28 & 16 & 4 \\ 7 & 28 & 49 & 28 & 7 \\ 4 & 16 & 28 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}, \quad \sigma = 1 \tag{2.2}$$

Equation (2.2) shows a discrete approximation to a continuous Gaussian kernel with $\sigma = 1$. Convolving this kernel with an image I of size $m \times n$ will produce output the smoothed image S:

$$S[m,n] = G_d[u,v] * I[m,n] = \sum_u \sum_v G_d[u,v] I[m-u, v-j] \tag{2.3}$$

The process of convolving an image with a kernel can be computationally complex at an order of $\mathcal{O}(mnuv)$. To reduce the complexity in the case for the Gaussian filter, one may exploit its inherent property of being separable. That is to say:

$$G_d^{sep} = f_d * g_d \tag{2.4}$$

For our example in Equation (2.2) it can be seen that:

$$G_d^{sep} = \frac{1}{17}\begin{bmatrix}1\\4\\7\\4\\1\end{bmatrix} * \frac{1}{17}\begin{bmatrix}1 & 4 & 7 & 4 & 1\end{bmatrix} \tag{2.5}$$

Convolving with image I can now be done in two separate steps, once for each new kernel:

$$S = I * G_d = I * (f_d * g_d) = (I * f_d) * g_d \tag{2.6}$$

This reduces the computational complexity to $\mathcal{O}(2mnu)$ when convolving the Gaussian kernel with an image. Gaussian filtering is not only important for reducing noise in an image, it is also a measure of scale. Convolving an image with a Gaussian kernel eliminates fine details (high frequencies) in the image, leaving hard edges to be smoothed.

Image courtesy: en.wikipedia.org/wiki/Lenna

A	**B**
C	**D**

A Original image. **B** $\sigma = 1$. **C** $\sigma = 3$. **D** $\sigma = 5$.

Figure 2.2: Gaussian blur's effect on image

Figure 2.2 shows the effect of applying the Gaussian filter with different standard deviations to an image, while retaining the kernel size.

2.2.2 Edge detection

Given a noise-free image, one may use another set of filters to find rapid intensity changes in the image. There are several approaches used for different applications. The simplest, and least time complex is the separable edge detector Sobel. Like the Gaussian smoothing filter, it employs convolution of a normalized kernel over an image. This kernel denoted E, is an approximation of the continuous derivative and is usually a fixed size 3×3 matrix.

In order to obtain both vertical and horizontal edges, this must be computed twice on the image, once for each kernel (E_x and E_y):

$$E_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad E_x^{sep} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} * \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \tag{2.7}$$
$$E_y = E_x^T$$

Image courtesy: en.wikipedia.org/wiki/Lenna

A	B

A Original image. **B** Sobel edge map.

Figure 2.3: Output of the Sobel edge detector

Figure 2.3 shows an example of applying the Sobel edge detector to an image. This figure presents an example of how filter convolutions extract features from an image. If this operation was done on a similar image of a face, there would certainly be similarities between the two, which can be taken advantage of during object classification. Applying filters of different kinds, some extracting edges, some extracting corners, is the fundamentals to any feature descriptor. These descriptors can then be matched to calculate the similarity between two images and score them accordingly. The next chapter presents the convolutional neural network (CNN). The CNN, as any other feature matching procedure is built on this same fundamental principle.

Chapter 3

An introduction to Convolutional Neural Networks

With the knowledge of filter convolutions from Chapter 2 it is time to investigate how employing a variable filter kernel can be used for object detection and classification in a image. This chapter introduce the fundamentals of a neural network, and how they are constructed. The idea of employing variable filter convolutions in a neural network is then described called the convolutional neural networks (CNNs). A presentation of how they are constructed, operate and trained are given. The general notation and derivation is inspired by Michael A. Nielsen's book "*Neural Networks and Deep Learning*" [2]. The deduction in Section 3.1.3 is largely based on a online lecture on the subject [4].

- Section 3.1 contains a breakdown of the components substantiating a CNN, starting with the neurons and layers constituting every neural network. The section aims to provide the reader with insight into the construction and training-procedure of a CNN. It also contains some of the important historical milestones in CNN for object classification.

- Section 3.2 introduces two CNN models which will be investigated further later in this thesis.

3.1 Convolutional Neural Networks

A convolutional neural network (CNN) is a neural network designed for the task of object classification for an image input. It is a deterministic approach to classification where features in the image are computed in *layers*. It differs from classical means of classification where the feature is extracted from the image i.e., Scale-invariant Feature Transform (SIFT) used to train a classifier such as k-Nearest Neighbor. The features of a CNN are hidden, and utilized for classification in one pipeline. Features in a CNN are generated

implicitly via one or several layers of filter convolutions. Each layer produces a set of abstract sub-images made from the input image. The optimal structure of the convolutional layers is problem dependent, and is typically determined iteratively through some cross validation procedure. In the broadest of sense, a CNN can be seen as a black-box, like shown in Figure 3.1. The user feeds the network with an image, and the network predicts the class-score for each class it has been trained on. What is contained in this black-box is often referred to as hidden layers, as the operations contained inside is not visible to the user.

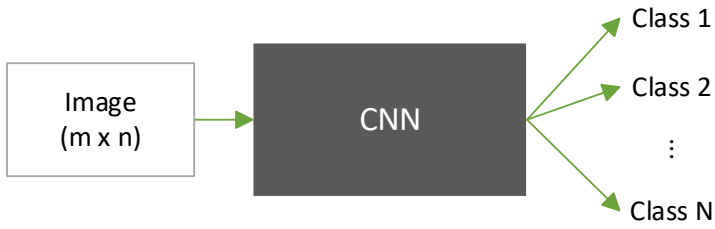


Figure 3.1: Black-box view of classification CNN

3.1.1 A brief history of CNNs

The CNN has acquired a large history over the relatively short time it has been researched. One of the first CCNs was the pioneering LeNet5 model from 1998 [23]. The name is derived from its creator Yann LeCun who developed this fundamental CNN for handwritten digit-recognition in images. By employing learnable filter kernels, pooling methods and induced non-linearities it decreased the error-rate over state of the art methods at the time [23].

In 2012, 14 years after LeNet was published, AlexNet was published [21]. This was a deeper and wider version of LeNet which was developed for a much harder problem. This deep CNN model was created to classify objects in images. This was previously not possible, but due to the exponential growth in computing power, this deep model was able to be trained on a GPU, drastically reducing the training time. It went on to win the annual Imagenet Large Scale Visual Recognition Challenge (ILSVRC) in 2012 under the team name ”SuperVision”. AlexNet sparked a huge interest in the computer vision community. In the following years, improvements were made to increase the robustness of these deep convolutional models, and the annual ILSVRC have been dominated by CNNs since then.

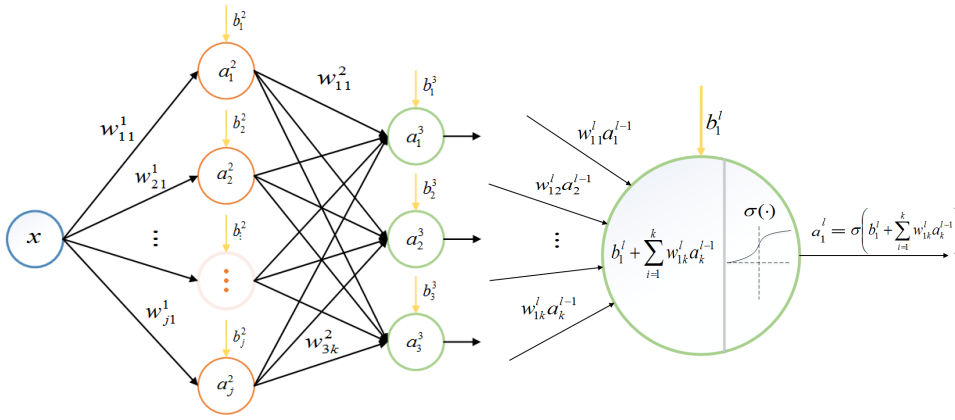
In 2013, a paper exploring why these CNNs performed so well was published by Matthew Zeiler and Rob Fergus [34]. This paper provided great insight into these previously abstract models. Based on the findings in the paper it also presents a shallow CNN with good object classification scores. From 2013 until the present day the most prominent and high-

scoring CNNs for object classification in images are the VGG, ResNet, and Inception-models [30],[18],[32],[31].

These last years have provided a large variation on the principle that Yann LeCun presented in 1998. Still, the fundamentals have stayed the same. Learnable filter kernels, pooling and induced non-linearities are employed in all the papers presented earlier in this section.

3.1.2 Neurons and Layers

To begin to understand the building blocks of a CNN we first look at the most rudimentary, yet the most important aspect of any neural network, namely *neurons* and *layers*. This gives basis for some of the definitions regarding any neural network.



A | **B** **A** Single layer, fully connected neural network. **B** A single neuron.

Figure 3.2: Simple feed-forward neural network

Figure 3.2 displays a simple, 1-dimensional neural network. Figure 3.2**A** shows a single input (blue) connected to a set of neurons (orange). These are then fully connected, which means that all neurons are connected to all output neurons (green). Figure 3.2**B** shows a single neuron.

Definition 3.1.1: Neuron: A single instance of one layer of a neural network. It receives one or several inputs and sum them together to produce a single output that is passed through an activation function.

Definition 3.1.2: Activation function: A function inducing non-linearity into an otherwise linear operation at the output of a neuron.

Definition 3.1.3: Layer: A set of neurons, each receiving different weighted varieties of the same input(s).

The input to the network in Figure 3.2A is denoted x and is a 1-dimensional input. In the sense of images, this might be viewed as a single, monochromatic pixel. In vectorized form, the input is weighted with the model-parameters $\mathbf{W}^1$.

Definition 3.1.4: Model-parameter: The term describes all parameters subject to optimization during training.

These weighted varieties of the input are passed into separate neurons, where they are summed and subject to an activation function $\sigma(\mathbf{W}^1 x + \mathbf{b}^1)$. The function descriptor σ is usually used to describe the logistic sigmoid activation function. In this thesis, all activation functions will be using this descriptor. The processed signal gets passed through a new set of weights $\mathbf{w}^2$ before entering the final output layer. In Figure 3.2B a single neuron is presented. The neuron takes in the weighted activations from the previous layer $\mathbf{W}^l \mathbf{a}^{l-1} + \mathbf{b}^l$ and sum them up. Let the linear operations of a single neuron l be defined as:

$$z_j^l = w_{jk}^l a^{l-1} + b_j^l \tag{3.1}$$

This operation can then be expanded to include all activations spanning the whole layer:

$$\mathbf{z}^l = \mathbf{W}^l \mathbf{a}^{l-1} + \mathbf{b}^l \tag{3.2}$$

This linear operation is passed to the activation function $\mathbf{a}^l = \sigma(\mathbf{z}^l) = \sigma(\mathbf{W}^l \mathbf{a}^{l-1} + \mathbf{b}^l)$. σ is a row-wise function such that $\sigma(\mathbf{v})_j = \sigma(\mathbf{v}_j)$. An additional parameter present here is the bias $\mathbf{b}^l$. This is a model-parameter that determines the horizontal alignment of the activation function. The details of the activation function is presented later in the chapter.

Model-Parameter	Description	Vectorized form
w_{jk}^l	Weight-parameter entering layer l at neuron j from neuron k	$\mathbf{W}^1 \in \mathbb{R}^{j \times k}$
b_j^l	Bias-parameter of layer l to neuron j	$\mathbf{b}^1 \in \mathbb{R}^{j \times 1}$
a_j^l	Activation function (output) of layer l from neuron j	$\mathbf{a}^1 \in \mathbb{R}^{j \times 1}$

Table 3.1: Model-parameters of a neural network

Table 3.1 summarizes the singular and vectorized representation of the inputs and output in a neural network layer.
Figure 3.3 shows a two-dimensional minimal example of a layer in a Neural Net. Here, a batch of (N) image-inputs are passed through 3 neurons (columns of the matrix).

Figure 3.3: Per-pixel weighted sum of image. Fully connected layer (transposed)

Definition 3.1.5: Batch: A stack of inputs of the same spatial dimension. Used to increase the amount of data for optimization before updating the model-parameters.

Using a batch of inputs, instead of a single input is desirable when training a neural-network, as it provides control over how often the model-parameters are updated. In order to do matrix multiplication on the pixels of the image in Figure 3.3, the rows are put end to end in a one-dimensional array, extending the second dimension to the batch size. It is evident that in this representation, the size of the bias matrix has to be predefined according to the batch size input. This would seem limiting from an implementation perspective, as it predefines the network to a single batch size. This is avoided, however, by defining the summand of the layer to be a broadcast. A broadcast dynamically changes the size of the bias matrix by replicating the first row N times. To have a batch of inputs extends the definition of model-parameters in Table 3.1 to include the extra dimension. Here we define $\mathbf{A^l}$ to be the batch output activations from a layer and $\mathbf{B^l}$ to be the biases.

Vectorized model-parameter
$\mathbf{W}^1 \in \mathbb{R}^{j \times k}$
$\mathbf{B}^1 \in \mathbb{R}^{j \times N}$
$\mathbf{A}^1 \in \mathbb{R}^{j \times N}$
$\mathbf{A}^{l-1} \in \mathbb{R}^{k \times N}$

Table 3.2: Extended model-parameters of a neural network

This allows for a representation of any layer to be given by

$$\mathbf{Z^l} = \mathbf{W^l A^{l-1}} + \mathbf{B^l} \tag{3.3a}$$

$$\mathbf{A^l} = \sigma(\mathbf{Z^l}) \tag{3.3b}$$

where we revert to the original definition of non-capitalized letters if $N = 1$.

3.1.3 Building a CNN

Section 3.1.2 shows how each input to each neuron has a unique weight. For large images, this amounts to a great deal of model-parameters for each layer. This becomes a problem under implementation, when the network undergoes rigorous training and testing. The training phase optimizes each weight and bias separately over a large amount of images. The sheer computing-memory needed to contain all model-parameters are infeasible when

creating a multi-layer neural network for large image inputs. To tackle this problem, several other conditioning layers are used in a CNN. There are in total four types of layers which CNN architectures are typically built upon.

Convolutional Layer: The core of the CNN is the convolutional layer. It consists of a set of filters, that are typically small in spatial dimension. As a input is passed through to a convolutional neuron, it slides (convolves) a filter kernel over the input to create the output. In reference to Section 2.2.2 describing the edge detector, this is analogous to convolving the Sobel operator over a very small image. Instead of having a static value for the filter-kernel however, they are regarded as model-parameters of a neural network. The output image of this convolution is called an activation map (edges in Sobel analogy). Each convolutional layer stacks the amount of filtered activation maps to pass on to the next layer in the network. What characterizes any respective convolutional layer are hyperparameters. A hyperparameter is a layer-fixed parameter. The term is used to distinguish these parameters from the model-parameters (weights and biases) and are not subject to any optimization. The hyperparameters are of fixed size, set by the user, or purely dependent on the input size. A convolutional layer can be described by 4 hyperparameters:

1. **K** number of filters
2. Spatial extent **F**
3. Stride **S**
4. amount of zero padding **P**

Figure 3.4 displays a convolutional layer. Here, as an example, a set of **K** filter kernels of spatial extent (**F**) 3 is convolved over a monochromatic image. There is no jump in the sliding of the kernel, as its center is displaced one pixel at a time, leading to a stride (**S**) of 1. In addition to the image input, there is added a 1 pixel wide zero padding to the brim of the image, resulting in a **P** of 1. The elements of the filter kernels are the weights in this case. Each filter down the depth **K** has unique weights, leading to **K** different activation maps. In the general case, the output volume can be described as follows: Given an input volume of size $[\mathbf{W_1} \times \mathbf{H_1} \times \mathbf{D_1}]$ to a convolutional layer, where $\mathbf{D_1}$ is the depth or the color channel ($\mathbf{D_1} = 3$ in the case of an RBG input image), it can be shown that the spatial size of the output is given by

$$\mathbf{W_2} = 1 + (\mathbf{W_1} - \mathbf{F} + \mathbf{2P})/\mathbf{S} \tag{3.4a}$$
$$\mathbf{H_2} = 1 + (\mathbf{H_1} - \mathbf{F} + \mathbf{2P})/\mathbf{S} \tag{3.4b}$$

in turn, the size of the convolutional layer output volume will be $[\mathbf{W_2} \times \mathbf{H_2} \times \mathbf{K}]$. The amount of neurons is given by $\mathbf{W_2} * \mathbf{H_2} * \mathbf{K}$. If every neuron output had a unique weight as in a fully connected layer, due to its nature this would make for a total of $\mathbf{W_2} * \mathbf{H_2} * \mathbf{K} * \mathbf{F} * \mathbf{F} * \mathbf{D_1}$ model-parameters.

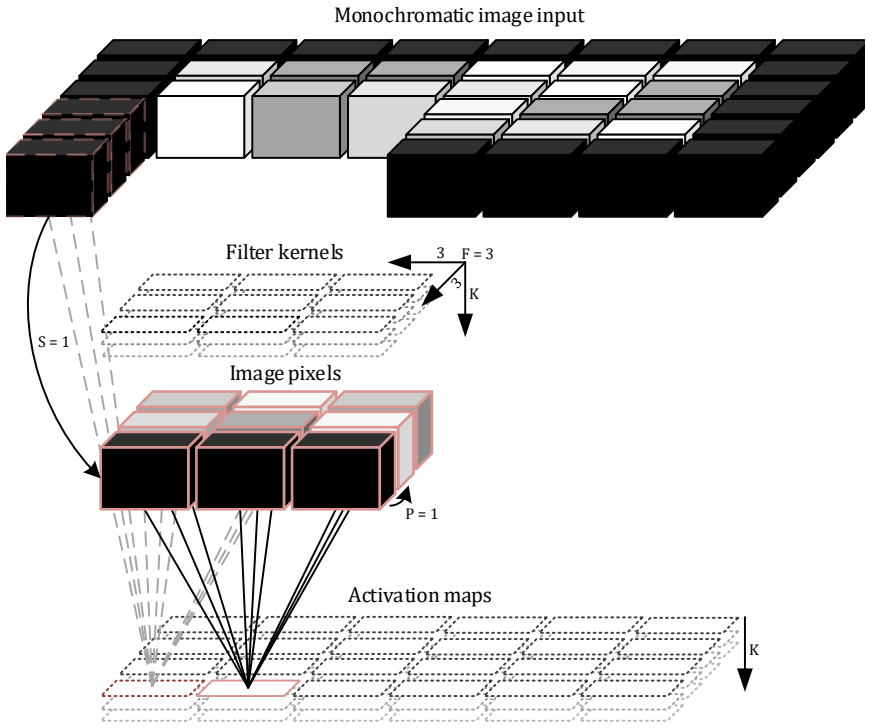


Figure 3.4: Convolutional layer

As an example, take the input image seen in Figure 3.4 of size $[6 \times 4 \times 1]$ to a convolutional layer with depth $\mathbf{K} = 3$. Setting $\mathbf{S} = 1$, $\mathbf{P} = 1$, $\mathbf{F} = 3$ it is evident that the spatial width of the output would be

$$\mathbf{W_2} = 1 + (6 - 3 + 2 * 1)/1 = 6$$
$$\mathbf{H_2} = 1 + (4 - 3 + 2 * 1)/1 = 4$$

the convolutional layer would consist of $6 * 4 * 3 = 72$ neurons where each neuron has a total of $3 * 3 * 1 = 9$ weights and one bias. This adds up to $9 * 72 + 72 = 720$ total model-parameters for a single layer, given a tiny, monochromatic picture as input. For larger inputs this results in a large amount of weights. CNNs reduce the necessity of unique weights by sharing the parameters between neurons throughout the depth of the layer. This approach leads to a total number of $\mathbf{K} * \mathbf{F} * \mathbf{F} * \mathbf{D_1}$ unique weights. In our example, this would lead to a total of $\mathbf{K} = 3$ unique sets of weights for a total of $3 * 3 * 3 * 1 = 27$ unique weights, or $27 + 3 = 30$ parameters including the bias. This example shows a 96% reduction in model-parameters compared to using a fully connected neural layer.

Rectified Linear Unit: Creating the activation map essentially means making a new, abstract batch of images from a batch of input images. The weighted filter-kernels used to create these images are tunable. Implicitly they can take on any real value, even negative. To determine if a neuron has been activated, this abstract image is subject to a activation function. A popular activation function used between layers in a CNN the later years is the Rectified Linear Unit (ReLU) [17]. It is a non-linear activation function that thresholds the output of the convolutional layer such that $\sigma(\mathbf{Z^l}) = max(0, \mathbf{Z^l})$, where $\mathbf{Z^l}$ is as defined in Equation (3.3a). This is the same step that is done in generic neural nets as is shown with the single input neuron in Figure 3.2**B**. The ReLU is specifically optimized for efficient computation, scale invariance and outputs sparse activations. This makes it ideal for image activations in a convolutional layer.

Pooling Layer: The pooling layer is a downsampling layer that is periodically inserted into the network to reduce the computations and the amount of parameters in the network. Downsampling the image leads to fewer activations from the previous convolutional layer. This is inevitable, but controlled using different downsampling techniques like max-pooling, which takes the largest value in each neighbourhood to preserve the activation. The pooling layer requires two hyperparameters: Spatial extent **F** and stride **S** as described earlier in this section.

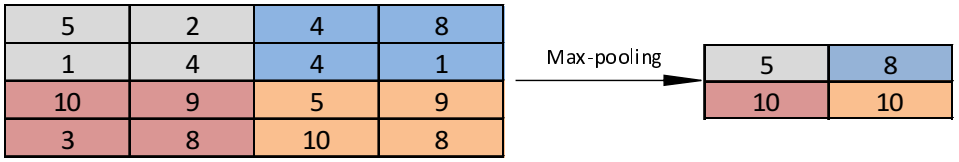


Figure 3.5: Example of max-pooling

Fully connected layer: The fully connected layer is presented in Section 3.1.2. Here a neuron has full connectivity to all activations in its previous layer, as seen in Figure 3.2. Each neuron is composed of a matrix multiplications between these activations as in Figure 3.3 to create one new activation. This is a typical final layer in classification, but is extremely computationally expensive.

3.1.4 Activations and classification scores

Activations are done at each stage of the CNN, inducing non-linearities into a otherwise linear system via the ReLU function [17]. The user however, only sees the activations done in the final fully connected layer, like what is shown in the black-box representation in Figure 3.1. These are the final classification scores. These classification scores typically lie in the range of 0 to 1. It is a measure of the relative activation strength of a class compared to the other classes.

To get a relative classification score for a set amount of classes, the output activations for each layer need to be related to each other in a meaningful way. Take a fully connected layer whose linear operation for a single image in the batch is given by $(\mathbf{z^l})$ consisting of (j) neurons with different sets of activations. The strength of a neuron in this layer will depend on the amount of activations relative to the other neurons in the layer. If one neuron holds all the activations, it will have a 100% classification score. One way of doing this is running the activations from each neuron through a normalized softmax function [9]. This function takes the vector $(\mathbf{z^l})$ of real values and outputs the vector $\mathbf{a}^l = \sigma(\mathbf{z^l})$. The values of the output lie in the range of $(0, 1)$ and add up to 1.

$$\sigma(\mathbf{z^l})_k = \frac{e^{z_k}}{\sum_{i=1}^{j} e^{z_i}} \quad \text{for} \quad k = 1, \dots j \tag{3.5}$$

The softmax function is a widely used activation function in CNNs. There are several alternatives, but the general idea of normalization to obtain a score is the same.

3.1.5 Putting it all together

We have seen the building-blocks that substantiate any CNN. These blocks can be put together, trained and tested. The optimal architecture is problem-dependent, and several articles exploring different configurations have been published [34],[30]. The general approach however, is a repetition of CONV-ReLU-Pool-CONV..., decreasing the spatial size of the activations throughout the network with the pooling layer.

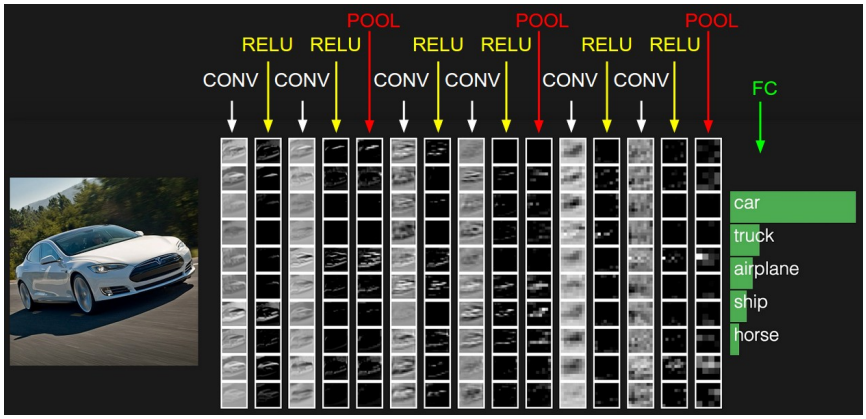


Figure 3.6: Example classification network, from [30]

Figure 3.6 shows an excerpt from the hidden layers as a CNN trained to classify objects in a image performs classification of a car. It consists of 6 convolutional layers, each spaced with a ReLU activation-layer, and then down-sampled by max-pooling at 3 instances. The final activation is computed by a fully connected layer with softmax activation.

3.1.6 Training a CNN

The process of determining the optimal model parameters for a CNN is called *training*. It is a supervised method, matching ground truth classes with the output predictions to minimize an objective function. This is done by passing annotated templates of the object that is to be detected. For each input, based on the outcome of the output-layer, adjust the weights and biases of the network to increase the correct probability for that image instance according to the ground-truth annotation. Repeating this procedure for a large amount of images, containing all classes allows the user to evaluate the strength of the CNN-architecture according to the convergence of the objective function . One way to go about training a Neural Net is by backpropagating the error function and then optimizing the weights and biases using gradient descent. For further reading on this visit [26]. This is one of the biggest challenges in employing neural networks, as a large amount of usually hand annotated inputs are necessary to create a robust classifier. For each template input these adjustments are made to all the model-parameters in all layers, which can be extremely time-consuming, even for modern CPUs. This section gives insight into the training procedure that is most commonly employed for convolutional neural networks.

Backpropagtion

During the forward-pass of a batch of annotated images through a CNN, the softmax activation function (3.5) outputs a normalized exponential distribution for the multi-class classification score $\mathbf{a}^L = \sigma(\mathbf{z}^L)$. Define the ground truth for each class to be contained in the vector:

$$\mathbf{y} = \{\mathbf{y} \in \mathbb{R}^{j\times 1}, 0 \leq y_j \leq 1\} \tag{3.6}$$

In pursuit of tuning the parameters to obtain better classification, the euclidean distance between the actual CNN output and the desired output is subject to a quadratic cost function:

$$C = \frac{1}{2N}\sum_{\mathbf{x}} ||\mathbf{y}(\mathbf{x}) - a^L(\mathbf{x})||^2 \tag{3.7}$$

This is an objective function, that we wish to minimize. The term that measures the penalty in this objective function is often called the loss-function. In this case it is the term $||\mathbf{y}(\mathbf{x}) - a^L(\mathbf{x})||^2$. The cost function is therefore, the sum of all loss-functions in the batch. N is the batch-size of training examples the CNN is subject to. A single sample from this batch is denoted $\mathbf{x}$. L is the total amount of layers in the network. The change of input variable is done to avoid confusion regarding intermediate fully connected layers not placed as the output-layer of the network. There are several cost function candidates tailored to different optimization purposes. To determine the effect that the weights and biases in the network has on the output of the cost function we need to compute the partial derivative:

$$\frac{\partial C}{\partial \mathbf{w}^l}, \frac{\partial C}{\partial \mathbf{b}^l} \tag{3.8}$$

To do this, an assumption must be made about the cost function, that is that the total cost function can be written as an average of all cost functions for each training image in the batch

$$C = \frac{1}{N} \sum_{\mathbf{x}} C_{\mathbf{x}} \tag{3.9}$$

where:

$$C_{\mathbf{x}} = \frac{1}{2} ||\mathbf{y} - \mathbf{a}^L||^2 \tag{3.10}$$

Under this assumption, backpropagation allows for the computation of

$$\frac{\partial C_{\mathbf{x}}}{\partial \mathbf{w}^l}, \frac{\partial C_{\mathbf{x}}}{\partial \mathbf{b}^l} \tag{3.11}$$

which outputs the partial derivatives for one training sample in the batch. The simple partial derivatives are then averaged over the whole set of training examples. As the vector $\mathbf{y^1}$ is of fixed size, $C_{\mathbf{x}}$ only regards the activations from the output-layer alone, with $\mathbf{y^1}$ as a parameter. To do this for each weight and bias in the network, define a small linear change to the activation functions to determine its effect on the outcome of the cost function. Equation (3.1) describes the linear operations for a single neuron in a layer. All activation functions throughout the depth of the network is perturbed such that they can be described by $\sigma(z_j^l + \triangle z_j^l)$. This change then propagates through the layers causing the overall cost to change by an amount of $\frac{\partial C}{\partial z_j^l} \triangle z_j^l$. Ddefine the error for a single training-example $\delta_j^{\mathbf{x},l}$ to be

$$\delta_j^{\mathbf{x},l} \equiv \frac{\partial C_{\mathbf{x}}}{z_j^l} \tag{3.12}$$

and subsequently in vectorized form δ^l is the error for layer l. Backpropagation gives a way of finding δ^l for each layer, and relating them to the quantities of interest in Equation (3.8). Four equations describe the backpropagation procedure, and is solved in the order presented:

$$\delta^{\mathbf{x},L} = \nabla_{\mathbf{a}} C_{\mathbf{x}} \circ \sigma'(\mathbf{z}^L) \qquad \circ : \text{dot product} \tag{3.13a}$$

$$\delta^{\mathbf{x},l} = ((\mathbf{w}^{l+1})^T \delta^{\mathbf{x},l+1}) \circ \sigma'(\mathbf{z}^l) \tag{3.13b}$$

$$\frac{\partial C_{\mathbf{x}}}{\partial \mathbf{b}^l} = \delta^{\mathbf{x},l} \tag{3.13c}$$

$$\frac{\partial C_{\mathbf{x}}}{\partial w_{jk}^l} = a_k^{l-1} \delta^{\mathbf{x},l} \tag{3.13d}$$

From these equations, it is evident why it is called backpropagation. Equation (3.13a) describes the error on the output-layer of the network. Here, $\nabla_{\mathbf{a}} C_{\mathbf{x}}$ describes the gradient whose partial derivatives are with respect to the output activations. $\sigma'(\mathbf{z}^L)$ measures the rate of change for the activation-function with respect to the linear operations. Equation (3.13b) describes the error in layer l with respect to layer $l+1$. Equation (3.13c) describes

the relationship between the error at any layer to the partial cost-rate for the bias. Equation (3.13c) describes the relationship between the error at any neuron in a layer to the partial cost-rate for the weights in the neuron. The last layer is described element wise as to not cause confusion with the notation. To see proof of the backpropagation procedure refer to [2]. Now, having a procedure to calculate the cost-function for an input, it is possible to apply Stochastic Gradient Descent (SGD) to optimize under the cost-function. For each layer in the network update the biases and weights according to

$$\mathbf{W}^l \rightarrow \mathbf{W}^l - \eta \sum_{\mathbf{x}} \delta^{\mathbf{x},l} (\mathbf{a}^{\mathbf{x},l-1})^T \tag{3.14a}$$

$$\mathbf{b}^l \rightarrow \mathbf{b}^l - \eta \sum_{\mathbf{x}} \delta^{\mathbf{x},l} \tag{3.14b}$$

where η is the step-size, or learning rate provided by the user. This is iterated upon until the system approaches a local minima. Given the nature of the cost-function, the weights and biases subject to optimization is only updated after the each batch of training-examples are evaluated. An important aspect of training a CNN, or any Neural Net is deciding the size of the batch. A small batch-size updates the weights and biases regularly, but may experience slow convergence, and is more susceptible to noise from outliers in the error-function. Large batches update the model-parameters at a slower rate, thus increasing the time of convergence for the cost-function. Choosing a large batch-size may in some instances not converge the cost-function at all.

3.1.7 Accuracy

Determining the accuracy of a CNN is done by calculating the precision/recall curve for the network under a set of annotated test-images. The precision represents the relevant instances of detection. The recall is the amount of relevant instances that are retrieved. This is described by four distinct output instances.

Full name	Abbreviation	Description
True Positives	tp	The amount of positive classifications relative to ground truth
False Positives	fp	The amount of positive classifications not identifying with ground truth
False Negatives	fn	The amount of ground truth not identified by classifier
True Negatives	tn	The amount of correct rejections by classifier relative to ground truth

Table 3.3: Relevance representations

Table 3.3 shows the four relevance representations, given a data set relative to a ground truth. The ground truth for classification is the annotation contained in the test image.

With these representations, the precision and recall of the CNN output can be computed by

$$p = \frac{tp}{tp + fp} \tag{3.15a}$$

$$r = \frac{tp}{tp + fn} \tag{3.15b}$$

with p denoting the precision and r the recall. Given the precision and recall, the Average Precision (AP) can be computed according to the VOC2012 documentation [13] as follows:

1. Sort the output data from the CNN with precision monotonically decreasing, by setting the precision for recall r to the maximum precision obtained for any recall $r' \geq r$.

2. Compute AP as the area under this curve by numerical integration.

This definition of AP will be used exclusively throughout this thesis. For multi-class classification, the term mean Average Precision (mAP) is used, which is simply the mean of the AP computed for all classes.

3.2 CNN-models for Classification

This chapter concludes with the investigation of two distinct CNNs made explicitly for classification of objects in an image.

3.2.1 Zeiler and Fergus model

The first model is known as ZF-net, proposed by Zeiler and Fergus in 2013 after attempting to increase robustness of CNNs by investigating the effects each layer of the network have on the output prediction [34]. It scored a 79% mAP on the VOC-2012 data set. The configuration proposed was well received and went on to win the annual Imagenet Large-Scale Visual Recognition Challenge (ILSVRC) in 2013.

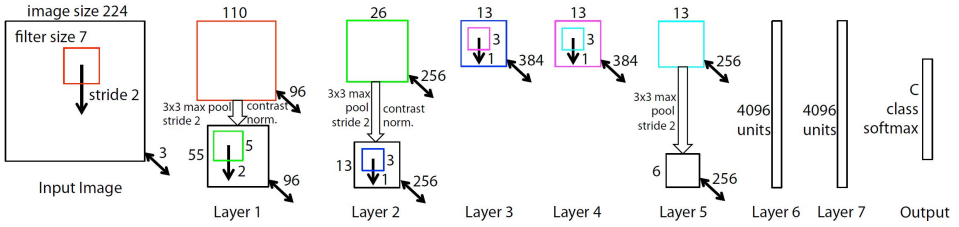


Figure 3.7: ZF-net model architecture [34]

Figure 3.7 shows the ZF-net architecture as proposed by Zeiler and Fergus. The input image, of arbitrary size is resized to fit a $[\mathbf{W_1} \times \mathbf{W_2} \times \mathbf{D_1}] = [224 \times 224 \times 3]$. The input is subject to $\mathbf{K_1} = 96$ filter kernels with spatial extent $\mathbf{F} = 7$ and stride $\mathbf{S} = 2$. Calculating the output spatial dimensions of this volume using Equation (3.4a) gives a output dimension of $[\mathbf{W_2} \times \mathbf{W_2} \times \mathbf{K_1}] = [109.5 \times 109.5 \times 96]$. The output volume has to be all integer valued. It is likely that an asymmetrical zero-padding has been added to the image. The most obvious solution is adding 1 pixel wide zero-padding to one of the sides for both the horizontal and vertical pixel span.

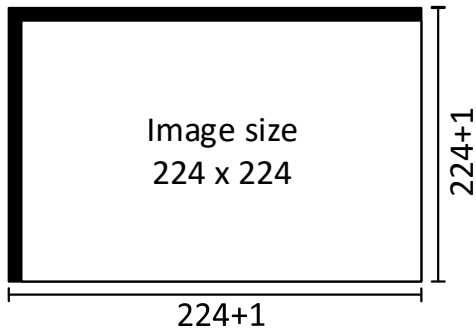


Figure 3.8: ZF-net possible input zero-padding

Figure 3.8 shows one possible solution, resulting in a correctly sized output volume $[\mathbf{W_2} \times \mathbf{W_2} \times \mathbf{K_1}] = [110 \times 110 \times 96]$. The output volume from the first layer is max-pooled to halve the spatial dimension as well as contrast normalized [21]. These activation maps are then passed through a ReLU layer which is not shown in the illustration. This operation is mirrored in the next layer, which is generated using $\mathbf{K} = 256$, $\mathbf{F} = 5$ and $\mathbf{S} = 2$ to generate a new set of activations. The third and fourth layer is pure convolutional layers with ReLU on the $[\mathbf{W_{3,4}} \times \mathbf{W_{3,4}} \times \mathbf{K_{2,3}}] = [13 \times 13 \times 384]$. Layer 5 again max-pools the activation map to reduce size. This greatly reduces the computational expenses for the two following 4096-neuron fully connected layers. The final probability distribution is a C-class softmax function as described in Equation (3.5), with C being the amount of classes. The model inhabits around 56 Million model-parameters.

3.2.2 VGG-16

The second model we explore is the deeper Visual Geometry Group (VGG) 16-layer model [30]. This model was proposed simply to increase accuracy compared to pre-existing models. It went on to win the ILSVRC-2014 challenge in the category of localization and classification reaching a mAP of 89.3% and 89% on VOC2007 and VOC2012 respectively. This was a rather astounding 10% increase in accuracy to the last years winner ZF-net, on the same data.

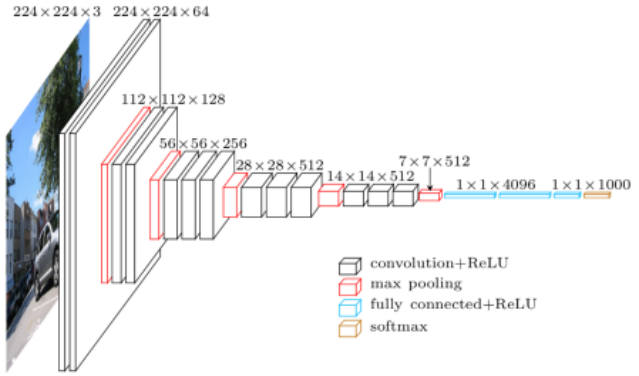


Image Courtesy: https://www.cs.toronto.edu/ frossard/post/vgg16

Figure 3.9: VGG-16 model architecture

Figure 3.9 shows the VGG-16 model architecture. The architecture, as ZF-net, resizes the input-image to fit a $[\mathbf{W_1} \times \mathbf{W_2} \times \mathbf{D_1}] = [224 \times 224 \times 3]$ window. In contrary to ZF-Net however, the convolutional layers of VGG-16 is not spatially reduced in size when subject to convolutional layers. This is done using square $\mathbf{F} = 3$ sized kernels with no skip $\mathbf{S} = 1$ and $\mathbf{P} = 1$ padding. The layers are still max-pooled at 5 instances to reduce the spatial extent. Again, similar to ZF-Net, they employ the ReLU activation for hidden layers, and C-class softmax for the probability distribution on the output. The model inhabits around 138 Million model-parameters.

3.2.3 Trade-offs

It is clear from the two models presented in this section that accuracy and model depth is closely related. The trade-off however, is the computation time for VGG-16, which is reported to be around 3 times longer than ZF-net. This seems reasonable as the model consists of about 3 times as many model-parameters. The VGG-16 model also requires a larger amount of computer memory to store a larger amount of feature-maps and model-parameters. During the training phase, these feature-maps are constantly subject to change, which means they need to be cached in RAM or GPU memory.

Chapter 4

Region-based Convolutional Neural Network

This thesis investigates a framework for detection and classification called Faster Region-based Convolutional Neural Networks (Faster R-CNN). The following sections explore the three dominant papers provided on this approach, leading up to the most recent framework for region based CNNs. There is a necessity to explore all three papers, as they heavily refer to its predecessors approach, and even using key parts described in these older works in the newer approaches.

- Section 4.1 provides a brief insight into R-CNN, the earliest work leading up to the state of the art framework used in this thesis.
- Section 4.2 looks at the faster approach to R-CNN, called Fast R-CNN.
- Section 4.3 is the most important section of this chapter. Here, a detailed deduction of the work in Faster R-CNN is provided. Details regarding the construction, loss and training aspects are investigated.

4.1 R-CNN

A Region-based Convolutional Neural Network is a term used for a single pipeline solving both a detection and classification problem. Here, regions of any size in an image are extracted, fed into a CNN and classified. In 2014 Ross Girshick et al. of UC Berkeley proposed a Regional Convolutional Neural Network (R-CNN) which achieved a 53.3% mAP on the VOC2012 test-set [16]. This was a 30% improvement over the previous best results [28]. This was achieved by first using selective search [33] to propose a number of regions in an image. The top 2000 region-proposals were then fed into a CNN performing classification. The key to the improved performance lies in the way that R-CNN is trained. The CNN is pre-trained on a large auxiliary data set not containing any bounding-boxes.

This is done using supervised training. The second stage is fine-tuning each domain in the training-set to achieve a class-specific AP.

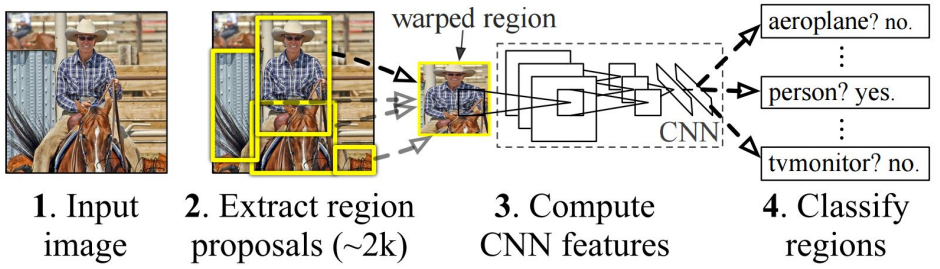


Figure 4.1: R-CNN pipeline [16]

Figure 4.1 shows the R-CNN pipeline, accepting an input image, subjecting it to selective search to extract regions, and classifying around 2000 of these regions.

4.2 Fast R-CNN

Fast R-CNN is a continuation of the work done by Ross Girshick in R-CNN [15]. In this paper, Girshick proposes a faster, more accurate approach to R-CNN, achieving a 66% mAP on the VOC2012 data set. This was achieved by creating a convolutional feature map of the image with all of its region-outputs from the selective search procedure. This allowed for pooling the regions of interest (RoIs) in one layer. By passing the pooled RoIs through two fully connected layers the pipeline outputs a softmax classification score, in addition to a bounding box regressor, according to the loss function

$$L(p, u, t^u, v) = L_{cls}(p, u) + \lambda[u \geq 1]L_{reg}(t^u, v) \tag{4.1}$$

where u is the ground truth class label, p is the predicted class, v is the ground truth bounding-box dimensions and t^u is the predicted bounding-box dimension. The term $[u \geq 1]$ evaluates to 1 for positive classes and 0 for negative. A box regression loss like this minimizes the initial bounding-box estimate to match the feature outputted by the CNN more precisely. The elements and composition of the loss-functions L_{cls} and L_{reg} are covered in greater detail in the subsequent section, where we look at the details surrounding the latest R-CNN publication, Faster R-CNN [25].

4.3 Faster R-CNN

The main part of this section is the latest iteration of the R-CNN approach, Faster R-CNN. This is the framework that will be evaluated and used in the pursuit of robust ship detection throughout this thesis. Faster Region-based CNN is developed at Microsoft Research by Shaoqing Ren et. al [25].

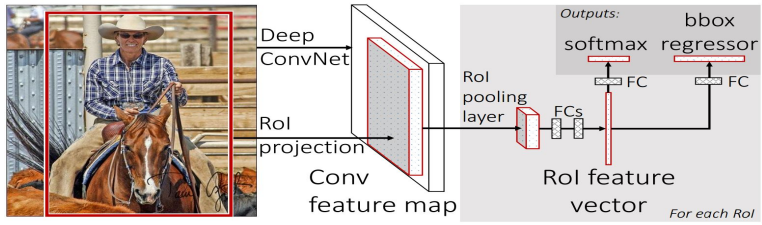


Figure 4.2: Fast R-CNN pipeline [15]

The major computational expense of Fast R-CNN in implementation was the use of selective search for region proposal [33]. Faster R-CNN eliminates the need for an external region-proposal algorithm by introducing what is called a Region Proposal Network (RPN). The RPN can be described as a fully convolutional network that takes an image input and outputs a correspondingly sized segmented output-map containing features in the image [24]. This feature map, and its corresponding RoIs are subject to a set of $k = 9$ rectangular *anchor-boxes* representing 3 scales and 3 aspect ratios. This is done by assigning each pixel in the output layer of spatial width $W \times H$ anchors, amounting to WHk total anchors for the feature map. Each spatial area covered by the RoIs is then passed to two sibling convolutional layers. The first convolutional layer outputs the classification score and the second the regression score with reference to these anchors.

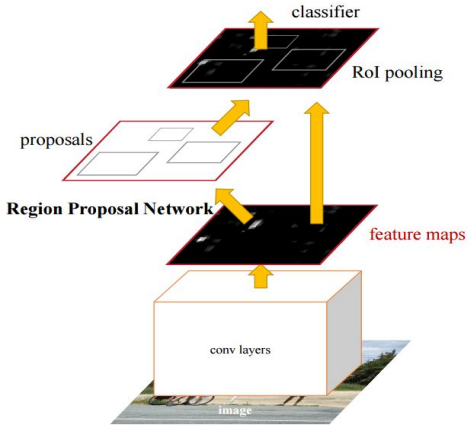


Figure 4.3: Faster R-CNN architecture [25]

Figure 4.3 shows an image being fed through convolutional layers to make activation-maps. The area of activation is then fed into the RPN to determine the strongest bounding-box. The spatial dimensions contained in these bounding-box proposals are fed into the classification layers where the classification score is obtained.
Training the RPN is done by assigning a binary class label for being an object or not to each region of interest. The positive labels are assigned two kinds of anchors: The highest

overlap with the ground truth bounding box and the RoI with the highest overlap with a Intersection over Union (IoU) with any ground truth box.

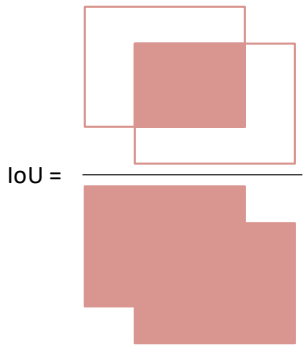


Figure 4.4: Visual depiction of Intersection over Union

The IoU is a ratio measure of bounding-box strength. It is simply the area of overlap divided by the area of union between two bounding boxes as shown in Figure 4.4. The classification loss and regression loss minimized according to the objective function:

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i, p_i^*) + \lambda\frac{1}{N_{reg}}\sum_i p_i^* L_{reg}(t_i, t_i^*) \tag{4.2}$$

Here, i is the anchor in a mini-batch and p_i is the probability of the anchor being an object. p_i^* is the ground-truth label and is 1 if the anchor is positive and 0 if not. t_i is the predicted bounding box coordinates and t_i^* is that of the ground truth. λ in the regression loss is a balancing parameter. To get a better understanding of how the objective function behaves we decompose the elements.

Classification score

Let's define the probability of the network outputting a true classification score as $p_{y=1} = \hat{y}$, leaving the opposite prediction to be $p_{y=0} = 1 - \hat{y}$. The equivalent notation for the annotated ground truth label is then given by $p^*_{y=1} = y$ and $p^*_{y=0} = 1 - y$. The distribution is then contained in $\{p \in \mathbb{R} : 0 \leq \hat{y} \leq 1\}$ and $\{p^* \in \mathbb{N} : 0 \leq y \leq 1\}$. The probability output is a classification score from a softmax activation function as defined in Equation (3.5) with $\mathbf{z}^{\mathbf{l}} = p$. In addition Faster R-CNN assigns these heuristics to the ground truth annotation:

$$p_i^* = \begin{cases} 1 & \text{if IoU} > 0.7 \\ 0 & \text{if IoU} < 0.3 \\ \text{excluded} & \text{if } 0.3 \leq \text{IoU} \leq 0.7 \end{cases} \tag{4.3}$$

Equation (4.3) gives a clear, dividing line between positive and negative predictions to minimize the chance of obtaining false objects at the edges of the ground truth bounding-

box. Faster R-CNN defines the classification loss to be a logistic loss function [25]. The loss function is defined as:

$$L_{cls}(p_i, p_i^*) = -log(p_i, p_i^*) = -[y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i)] \tag{4.4}$$

It is evident from Equation (4.4) that the function is minimized and penalized under two instances:

1. Minimized if there is a strong overlap between the anchor and a ground-truth bounding box.

2. Penalized if there is a strong prediction on a false ground-truth bounding-box.

Box regressor

The second part of the RPN objective function is the box-regression loss. This is described as a smooth absolute loss over all vectorized bounding-box coordinates. The bounding-boxes are vectorized as follows

$$\begin{array}{ll} t_x = (x - x_a)/w_a, & t_y = (y - y_a)/h_a \\ t_w = log(w/w_a), & t_h = log(h/h_a) \\ t_x^* = (x^* - x_a)/w_a, & t_y^* = (y^* - y_a)/h_a \\ t_w^* = log(w^*/w_a), & t_h^* = log(h^*/h_a) \end{array} \tag{4.5}$$

where x, y, w, h denote coordinates, width and height. The variables x, x_a, x^* are the predicted, anchor and ground truth bounding boxes successively.

$$p_i^* L_{reg}(t_i, t_i^*) = p_i^* R(t_i - t_i^*) \tag{4.6a}$$

$$R(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases} \tag{4.6b}$$

The regression loss is only activated for true samples, having a IoU with the ground truth of over 0.7. It penalizes the objective function via the loss to either increase or decrease the size of the predicted bounding-box compared to the 9 anchors' spatial difference to the ground truth.

4.3.1 Training Faster R-CNN

Faster R-CNN present three training solutions, *Alternating training, Approximate join training and Non-approximate join training*. Detailed description of these are for the reader in the Faster R-CNN article [25]. All experiments in Faster R-CNN are done using alternate training. This training solution is also used for all experiments presented later on in this thesis. This section gives a description of *Alternate training*.

Alternate training treats the training as a four step process, alternating between training the RPN and the fast R-CNN classifier.

Figure 4.5 depicts the training procedure as described in Faster R-CNN. The first step is training the RPN. Here, the images in the training set are fed to the network in batches of 256 images at a time. The weights and biases of the network are updated after each batch. This is iterated over 80.000 image-batches. The learning-rate η decreases from $\eta = 0.001$ to $\eta = 0.0001$ after 60.000 iterations. The two first steps of training initialize the weights of the network using a pre-trained model. The paper does not explicitly define the classes or images used to train these pre-trained models, other than that they are trained using images from the imagenet repository [11]. In the Faster R-CNN repository, two pre-trained models are provided, one for VGG-16 and one for ZF [29]. The layer-specific weights for the RPN in the first step is initialized at random from a zero-mean Gaussian distribution. All layers of the network is then trained end-to-end on the training-images. The RPN training outputs a large amount of regions for each image. These regions are subject to Non-Maximum Suppression(NMS) which only stores the top 2000 regions for any given image to be used for classification. A region is rejected under the criteria that it has a IoU overlap with a higher scoring region larger than a learned threshold [16].

The second step in the training procedure is training fast R-CNN on the image regions proposed by the RPN. At this stage, the two networks do not share any layers. Fast R-CNN is initialized from the same pre-trained model as the RPN at this stage. It is then trained end-to-end, batching 2 images and 64 random regions at a time. Fast R-CNN is by default trained over 60.000 iterations, decreasing the learning-rate from $\eta = 0.001$ to $\eta = 0.0001$ after 30.000 iterations.

The third step is training the RPN again. This time it is initialized from the weights from step 2, while the RPN specific layers are initialized from the optimized parameters from step 1. At this stage, the training only fine-tunes the RPN specific model parameters, keeping the stage 2 parameters static. The learning-rate and iterations is the same as for stage 1.

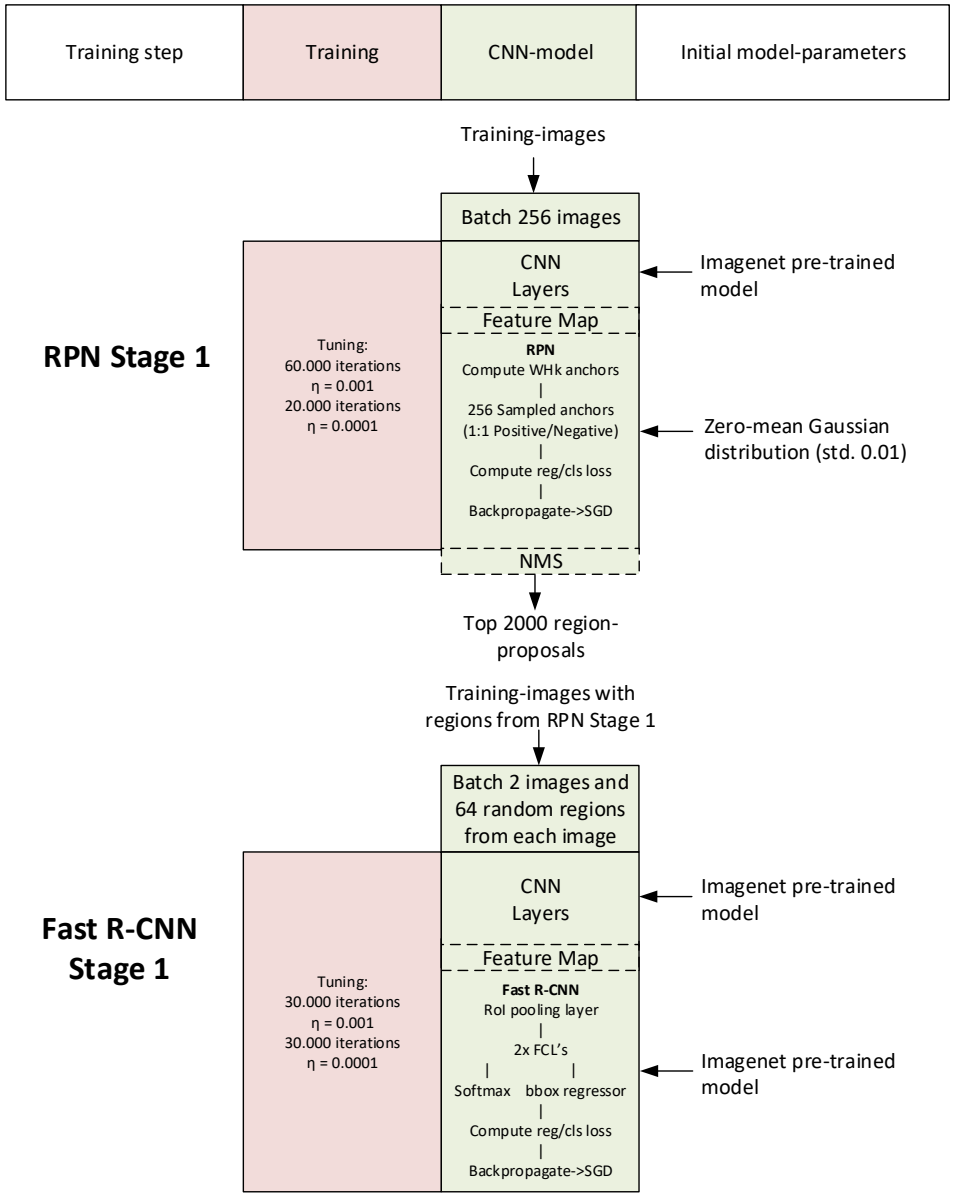
The last training-step is the same as the third in terms of training, keeping the shared layers static, while fine-tuning the Fast R-CNN layers. The learning-rate and iterations is the same as for stage 2.

When training is complete, the RPN and Fast R-CNN layers share model-parameters in the common ”Shared CNN Layers”. At this stage they are merged together to form one unified network. This network takes one single image and proposes regions from the RPN, classified with Fast R-CNN. The RPN is ran end-to-end, taking a single image, proposing regions and applying NMS to reduce the amount of RoIs. These RoIs, along with the feature-map at the last shared CNN layer is fed into the Fast R-CNN layers for classification. This eliminates the need to run the image through the shared convolutional layers twice.

Testing

Testing Faster R-CNN is a task of running single images through the trained network and computing the precision/recall and its associated AP for each class and mAP for the all classes according to what is described in Section 3.1.7. During testing it was shown that

keeping 2000 regions after NMS actually gave a poorer outcome during classification. In the testing-phase, 300 regions were shown to give a slight increase in performance when sharing the model-parameters between the RPN and Fast R-CNN layers [25].


Training step
Training
CNN-model
Initial model-parameters
Training-images
Batch 256 images
CNN Layers
Imagenet pre-trained model
Feature Map
RPN Stage 1
Tuning:
60.000 iterations
η = 0.001
20.000 iterations
η = 0.0001
RPN
Compute WHk anchors
256 Sampled anchors
(1:1 Positive/Negative)
Zero-mean Gaussian distribution (std. 0.01)
Compute reg/cls loss
Backpropagate->SGD
NMS
Top 2000 region-proposals
Training-images with regions from RPN Stage 1
Batch 2 images and 64 random regions from each image
CNN Layers
Imagenet pre-trained model
Feature Map
Fast R-CNN Stage 1
Tuning:
30.000 iterations
η = 0.001
30.000 iterations
η = 0.0001
Fast R-CNN
RoI pooling layer
2x FCL's
Softmax
bbox regressor
Imagenet pre-trained model
Compute reg/cls loss
Backpropagate->SGD

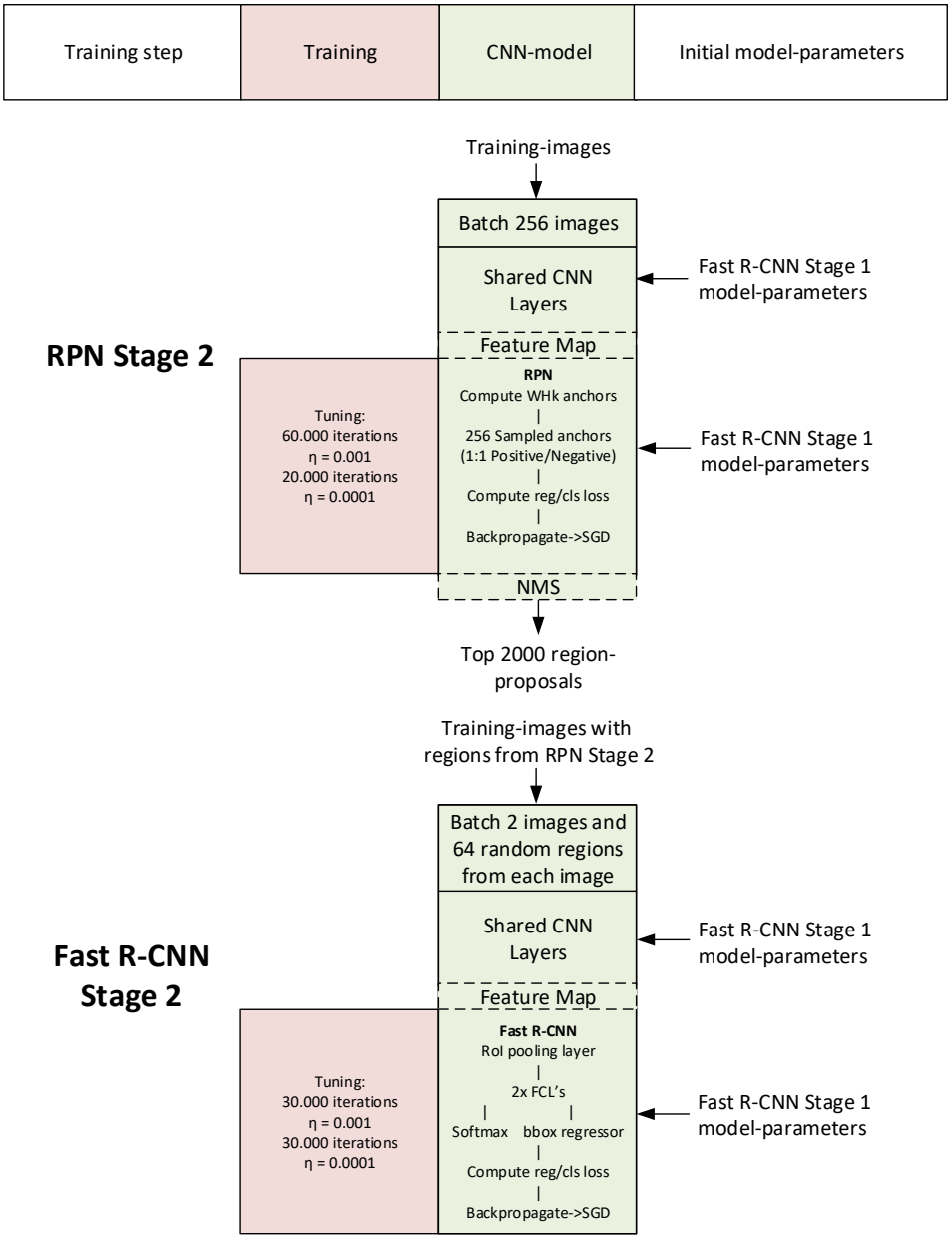


Figure 4.5: Faster R-CNN 4-step alternate training

Part III

Method

Chapter 5

Datasets

Chapter 4 briefly introduced the imagenet and Visual Object Challenge (VOC) data sets. There are some inherent differences in the images contained in these data sets. The experiments conducted later on in this thesis will be utilizing these data sets for the purpose of training an object detector. This chapter presents the various data sets and the construction of a data set specifically designed for the ASV application.

- Section 5.1 presents the 2007 and 2012 Visual Object Challenge data sets. A small discussion regarding the relevance of the data is presented throughout the section.
- Section 5.2 presents images pulled from the image database imagenet. These are images chosen for their relevance to the ship detection objective.
- Section 5.3 concludes the chapter with the presentation of a custom data set made for this thesis.

5.1 Visual Object Challenge data sets

Faster R-CNN comes with pre-trained ZF and VGG-16 models. These models have been trained on the VOC2007 and VOC2012 data sets, as well as a combination of the two (VOC0712). These data sets are part of the annual challenge, and are changed and expanded upon every year [13]. The VOC data sets come partitioned in 4 distinct image sets.

In Table 5.1 the four image sets are described. All image sets contain ground-truth bounding-boxes of the 20 classes that the contest requires researchers to evaluate their CNN over. There is however a large variation in the amount of objects for each class throughout the image sets. The contents of these data sets have direct impact on the robustness expected from the output of the fully trained model. It is therefore desirable to have a closer look at the images contained here.

train	Training images used to minimize the objective function and update model-parameters.
val	Validation images for minimizing overfitting. Same as a testset, but parsed periodically in the training regime to verify accuracy outside the training images.
trainval	A combination of the images contained in the training and validation data sets.
test	Test images used to evaluate the mAP of the CNN after training.

Table 5.1: VOC main image sets

	train		val		trainval	
	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

Table 5.2: VOC2007 image and object specifications

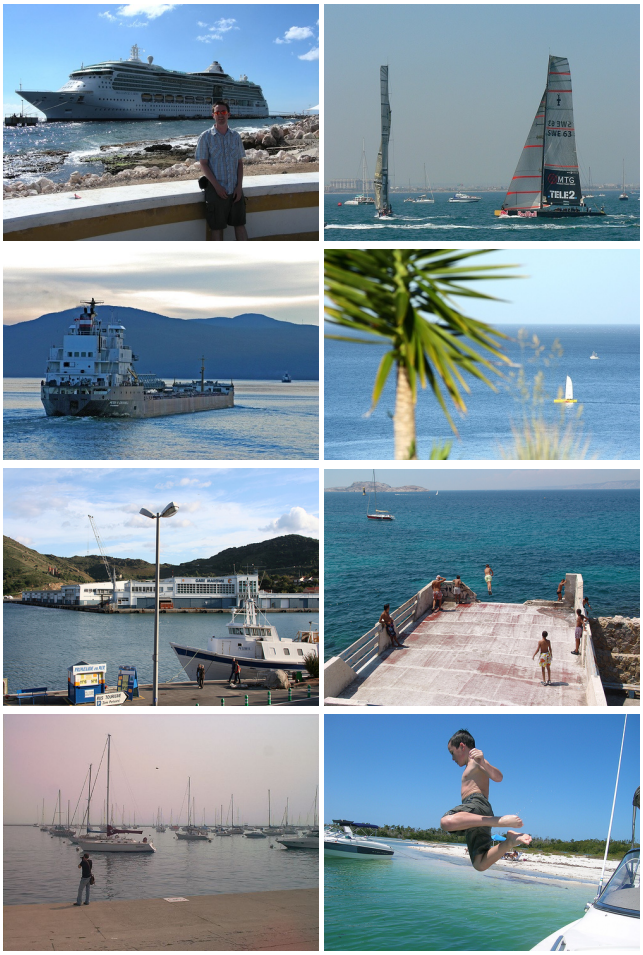
	train		val		trainval	
	img	**obj**	**img**	**obj**	**img**	**obj**
Aeroplane	327	432	343	433	670	865
Bicycle	268	353	284	358	552	711
Bird	395	560	370	559	765	1119
Boat	260	426	248	424	508	850
Bottle	365	629	341	630	706	1259
Bus	213	292	208	301	421	593
Car	590	1013	571	1004	1161	2017
Cat	539	605	541	612	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

Table 5.3: VOC2012 image and object specifications

Table 5.2 and Table 5.3 displays the total amount of images contained in the VOC2007 and VOC2012 training and validation data sets. From the tables it is clear that the class of interest in this thesis, boat, is rather poorly represented relative to the other classes. 4.2% of the images, and 2.8% of the object bounding-boxes contained in both training-image sets belong to the class. On the other hand, the person class, which is the most represented class, hold 36% of the images, and 33% of the objects in the same image sets.

This thesis considers the VOC2007 test-set for benchmarking. These are images used for calculating the precision and recall, as well as the mAP discussed in Section 3.1.7. The test-set contains 4952 annotated images that are unavailable during training. There is no information about the amount of images containing the different classes and their bounding-boxes, as is presented for the training set in Table 5.2.

In Figure 5.1 some excerpts from the VOC2007 and VOC2012 data sets are displayed. These images do not contain each class explicitly. For instance, in Figure 5.1A a person is posing in front of a docked ship. In some images, as seen in Figure 5.1E, the boat is occluded at the far edge of the image, discontinuing the features. If one were to draw bounding-boxes around these objects, information would certainly be lost. During training, using images where features from one class overlap the other could possibly induce a certain degree of error, miss-classifying the one for the other.



A	B
⋮	
G	H

A-D Excerpts from the VOC2007 data set. E-H Excerpts from the VOC 2012 data set. All images including the boat class.

Figure 5.1: Image samples from the VOC2007 data set

5.2 Imagenet

Imagenet is a large scale hierarchical image database created at Princeton University [11]. The aim of imagenet is simply to encompass the largest amount of annotated images for a vast amount of objects. The hierarchy of imagenet is largely a symantical tree-structure, where each branch is a sub-category of the parent. These branches are called synonymous sets, or *synsets*. The categorization of these synsets is based on a earlier Princeton database project called wordnet [10]. Wordnet is an English lexical database, where the children in the tree are synonymous with their parents. Imagenet expands on wordnet, including annotated images for these synonyms. Upon release, imagenet contained images in around 10% of the wordnet synsets, with 3.2 million images. At this time, imagenet has expanded to contain 19% of the synsets with 14 million images. There is a key difference between images from imagenet and those from the VOC image sets. Imagenet images are only annotated with its respecitve synset class identification. If any other object not belonging to that synset is in the image, it will be regarded as background. On the other hand, a VOC image can contain multiple classes.

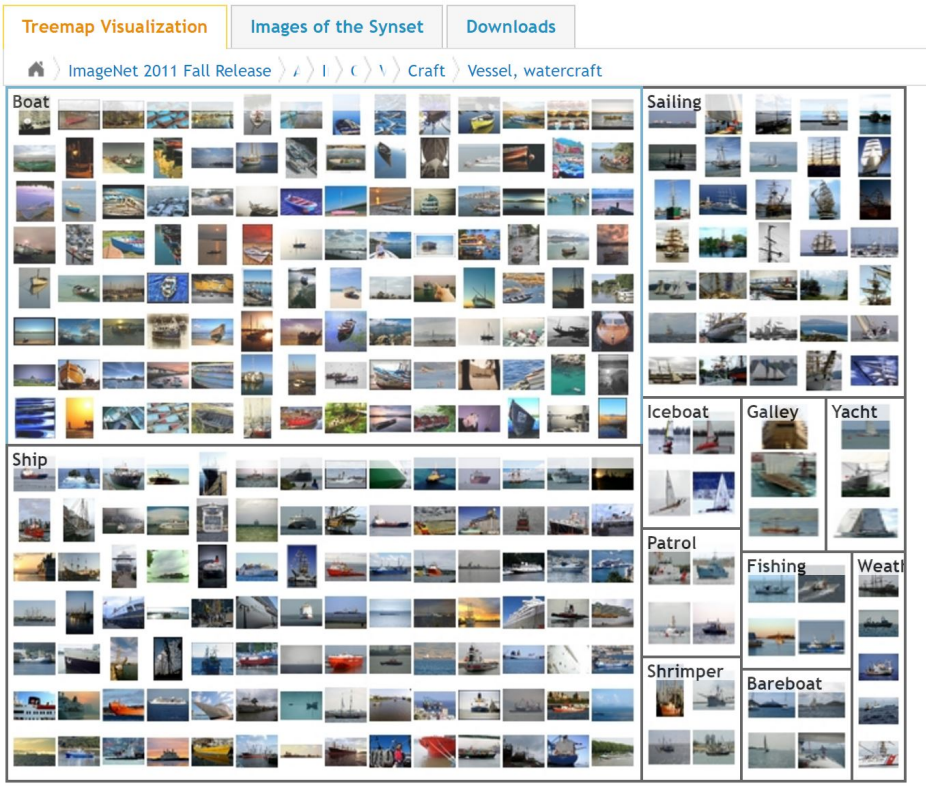


Figure 5.2: The children of synset *Vessel,watercraft* from http://image-net.org/.

Name	Synset-ID	Num. annotated images
Boat	n02858304	199
Ship	n04194289	1022
Sea Boat	n04158807	199
Watercraft	n04530566	239
Total		1659

Table 5.4: Imagenet synsets in thesis

Figure 5.2 display an instance of the imagenet tree. Here, the parent *Vessel,watercraft* is shown to contain children image sets such as *boat, ship* and *yacht*. The images in these synsets are publicly available images, pulled from image hosting webpages such as www.flickr.com. They are then hand annotated to match their respective synsets.
The imagenet synsets used in this thesis are shown in Table 5.4. All the synsets shown here contain a total of 1893 bounding-boxes. These synsets were chosen as they largely represent boats and ships at sea, with minor contamination from other classes. Combining these four synsets also give a large variety of different water vessels. Good bounding-boxes on these images will obtain a large amount of sea, terrain and mountains as negative examples.

A	**B**	**C**	**D**
E	**F**	**G**	**H**

A-B Excerpts from the boat data set. **C-D** Excerpts from the ship data set. **E-F** Excerpts from the sea boat data set. **G-H** Excerpts from the watercraft data set.

Figure 5.3: Image samples from the VOC2007 data set

Figure 5.3 shows some excerpts from the imagenet synsets. These imagenet synsets contain little noise from other classes, while containing the class to be detected with little to no occlusion. This is not true for all the images in the set, but in comparison to the VOC images, the general features of the objects to be detected are better contained in the image. One example of contamination can be seen in Figure 5.3**F**, where a person is included in the scene.

5.3 Data collection

On March 31st 2017, an expedition was conducted to gather data of ships with different sensors, including RADAR, AIS and camera. The expedition consisted of driving a boat around the Trondheim harbor and at open sea to gather data in the different scenarios that an ASV may encounter. The captured footage mainly consists of ships at rest in the harbor, or ships at sea in transit. The weather conditions were mostly sunny, with some clouds. There was no precipitation throughout the expedition.

Camera	Abbreviation	Resolution	Position
GoPro HERO 5 Black	c_1	1920×1080	Above helm, facing bow
GoPro HERO +	c_2	1920×1080	Off the starboard quarter
GoPro HERO +	c_3	1920×1080	Off the port quarter

Table 5.5: Camera specifications

Three cameras were used to capture images with different field of views. Table 5.5 shows the specification of the three cameras attached to the boat. During the expedition, all three cameras were triggered simultaneously with a GoPro smart remote.

A Camera c_1. **B** Camera c_2. **C** Camera c_3.

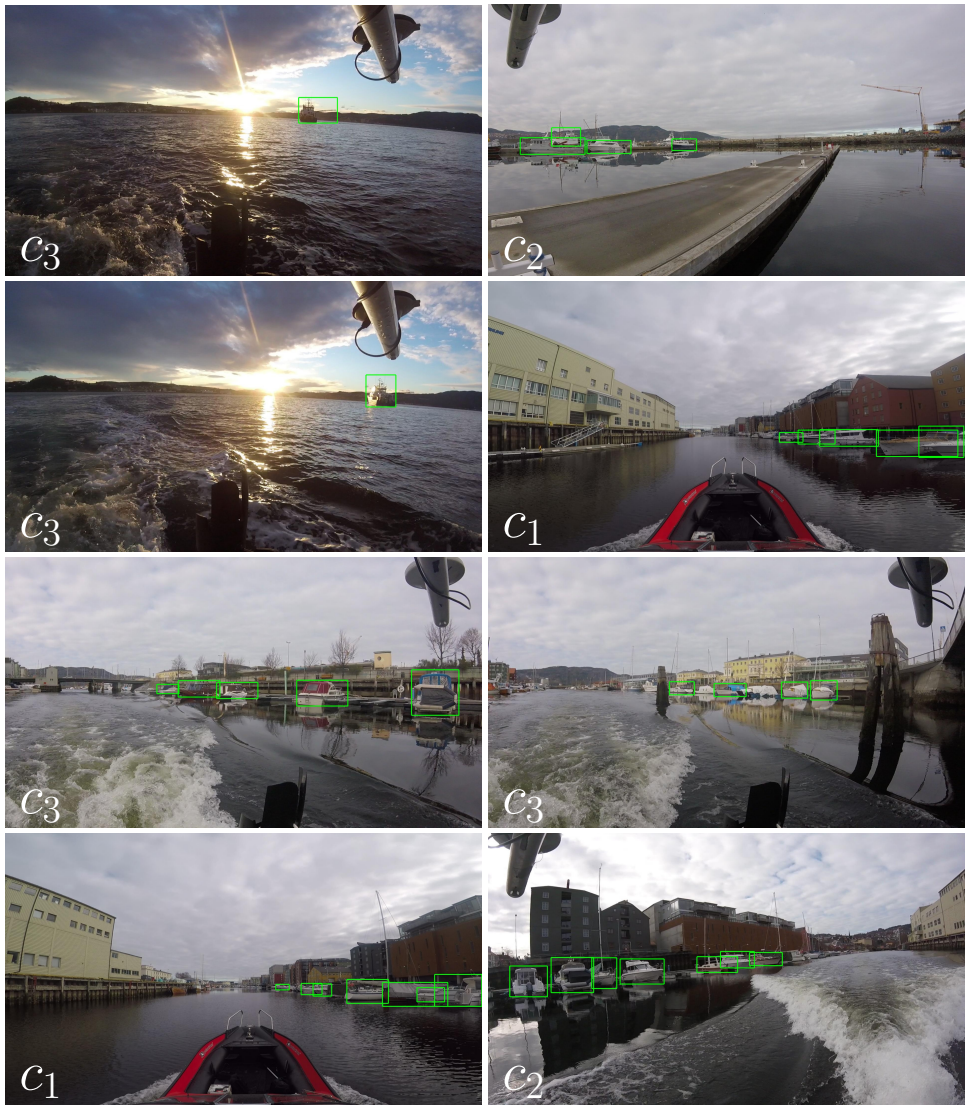
Figure 5.4: Image samples displaying the 3 camera's view

Figure 5.4 shows images captured at the expedition. The forward facing c_1 camera captures parts of the host-vessel in its image, while the backwards facing cameras capture a part of the GPS antenna. Samples from these videos were used to create what will be

denoted as the custom data set. This image set consists of 177 annotated images from this expedition and is split into two distinct data sets, one for training and one for testing. The data sets are split with a small majority contained in the training-set. This can be seen in Table 5.6. The custom data set is not comparable to the VOC or imagenet data sets in terms of size. It does, however, contain a large amount of ground-truth bounding boxes per image, with an average of 4.7 boxes per image. VOC2007 and VOC2012 in comparison, inhibit an average of 1.7 boxes per image. The aim for these data is to provide data for training and testing in close relation to each other. When it comes to evaluating the training and testing procedures later on in this chapter this will be a benchmark for creating a more local detection model. A local model in this sense is a model where images are collected on a predefined path, e.g., for an autonomous ferry. These images are then used to train the network. The hypothesis here is that these images will eliminate some of the false positives arising from miss-classifying the background or other objects in the images.

Dataset	train	test	gt-boxes
Custom	100	77	834

Table 5.6: Custom data set partitioning



A	B
⋮	
G	H

A-H Sample images from the annotated data set. Ground truth boxes in green.

Figure 5.5: Image samples from the custom data set with bounding boxes.

Chapter 6

Implementation

The previous part of this thesis aimed provide insight into the theory surrounding CNNs and the object detection framework Faster R-CNN. This chapter presents the implementation details of running and training CNN models within this framework.

- Section 6.1 gives an overview of the hardware and software used for implementing Faster R-CNN in this thesis.
- Section 6.2 presents the Faster R-CNN script, Caffe C++ library and CUDA for parallel computing.
- Section 6.3 consider some of the implementation aspects to Faster R-CNN. This includes the comprehensiveness of documentation and challenges of using the framework.

6.1 Overview

This section presents the hardware and software used for training and testing the CNN models in the experiments presented later on in this thesis.

	Computer 1 ($Comp_1$)	Computer 2 ($Comp_2$)
Processor	Intel(R) Core(TM) i5-4690K @ 3.50GHz	Intel(R) Core(TM) i5-4690K @ 3.50GHz
Graphics unit	Nvidia GeForce 1080 @ 1607MHz	Nvidia TITAN Xp @ 1582MHz
GPU Memory	8GB	12GB
Memory (RAM)	8GB	8GB
Architecture	64-bit	64-bit

Table 6.1: Hardware specifications

Operative system	Windows 10 Home
MATLAB version	R2014a
Parallel computing platform	Nvidia CUDA 8.0
IDE	Microsoft Visual Studio 2015

Table 6.2: Software specifications

The decision to use an older version of MATLAB was done to initially recreate the implementation done as it was in the repository for Faster R-CNN [29]. In later stages it proved sufficient in the process of training and evaluating self-trained networks. Table 6.1 describes two separate computer builds which are equal in every aspect except for the graphics unit used. Comp_2 was built to provide sufficient GPU memory for the deeper VGG-16 model, when Comp_1 proved insufficient for the task.

6.2 Faster R-CNN

Faster R-CNN is provided under the MIT-license. The license and its condition can be found at the provided repository [29]. The repository provided for the object detection framework is a MATLAB implementation of what is described in the original paper [25]. A python implementation is also available. Even though the program uses MATLAB, most computations are done in C++, which gives access to parallel computing using CUDA. The interface that allows MATLAB scripts to run C++ headers and subroutines is called a MATLAB ExeCutable (MEX). Faster R-CNN is compiled in Caffe, a C++ deep learning framework from Berkeley Vision [19].

6.2.1 Caffe

Caffe is a C++ deep learning framework utilizing Nvidia CUDA for parallel computing. It is an in-development framework with a large amount of contributors. This thesis will not directly modify the C++ Caffe framework during implementation, but its structure is important for understanding the Faster R-CNN framework.

Caffe was made to address the lack of *"off-the-shelf deployment of state-of-the-art models"* [19]. This is done with an all incorporated toolbox which allows researchers to build, train and ultimately distribute the training-parameters alongside their articles for peer-review. The framework comes with both Python and MATLAB bindings for constructing and training neural networks. The framework includes classes for reading data and the most notable functions for computing any Neural Network spanning from simple fully-connected networks to more advanced networks such as CNNs and Recurrent Neural Networks (RNNs). Caffe is made with consideration for the multi-platform bindings. This is evident from the way a Caffe NN-model is universally built and trained. All layers and the solver are defined using text-based file extensions named .PROTOTXT.
Figure 6.1 shows how a convolutional layer is defined in Caffe using the prototxt. Upon training or running a NN, Caffe will load the appropriate prototxt and the NN is defined

```
layer {
      name: "conv1"
      type: "Convolution"
      bottom: "data"
      top: "conv1"
      param {
            lr_mult: 0.0
      }
      param {
            lr_mult: 0.0
      }
      convolution_param {
            num_output: 96
            kernel_size: 7
            pad: 3
            stride: 2
            weight_filler {
                  type: "gaussian"
                  std: 0.01
            }
            bias_filler {
                  type: "constant"
                  value: 0
            }
      }
```

Figure 6.1: Caffe convolutional layer

accordingly. Similarly, the solver can be defined, incorporating the step size and the number of iterations the NN is subject to during the training and testing phase. In employing Caffe, there are two outputs to consider: Under training, Caffe updates the model file denoted .CAFFEMODEL with the state of the weights and biases at each layer of the network. In addition Caffe outputs the state of the solver. This is done alongside the updated model-parameters. This output is denoted .SOLVERSTATE. The solverstate contains information required to continue training should it have exited at any stage during the training-process, such as which iteration it is to continue from.

6.2.2 Parallel Computing

Employing a CPU to do all convolutions, activations and general calculations during training is a tall order, as the way it is designed only allows for the computation of a single operation at a time, per core. Optimizing the error over several million model-parameters for tens of thousands of iterations will take a large amount of time, and is not feasible. The way neural networks are designed allows for multiple sub-routines to be ran in parallel, as the output of one neuron in a layer is independent from the output of another. A GPU in comparison to the CPU is designed with a much larger amount of computation-cores. For instance, the Nvidia GeForce GTX 1080 which is used to provide some of the results in this thesis is designed with 2560 cores. These cores can each process simple sub-routines simultaneously. This greatly reduces the training-time for a deep CNNs. Faster R-CNN and the Caffe framework utilizes Nvidia CUDA for parallel computing.

CUDA

CUDA is the parallel computing platform developed by Nvidia. It envelopes libraries such as Basic Linear Algebra Subprograms (BLAS), Integrated Performance Primitives (IPP), which include Computer Vision and Image Processing operations as well as Fast Fourier Transform (FFT). These CUDA libraries replace the standard BLAS, IPP and FFT libraries found in most programming languages to one that allows users to perform these operations in parallel using a Nvidia GPU. It supports languages such as C/C++, Python and Fortran.

6.3 Implementation Aspects

Looking at the repository provided for Faster R-CNN, it is presented as a plug-and-play MATLAB framework, ready to be utilized for testing and training [29]. It contains detailed instructions for hardware and software requirements for both training and testing. This section contains some of the implementation aspects that any new user of the Faster R-CNN should keep in mind during setup, testing and training of the framework. The implementation aspects in this section only concern the MATLAB implementation of Faster R-CNN. Any user of Python should refer to the Python implementation of the framework which is linked in the repository. This section also provides some of the challenges faced during implementation.

6.3.1 Documentation

This section covers the documentation aspects of the different frameworks.

Faster R-CNN

Faster R-CNNs complete documentation consists of applying the knowledge gained from the Faster R-CNN article [25] to the instructions provided in the Faster R-CNN repository [29]. The code is nicely structured and well commented, but there is a lack of comments on all functions not in the high-level ”user intended” scripts. Common problems with the implementation are mostly covered in the issues section for the repository. A good way to unravel the highly nested code is to use the repository search. This provides all uses of a given script/function or variables. The lack of official proper documentation is unlikely to be addressed at any time. At the time of writing, the last official commit to the repository was 9 months old.

Caffe

Although a Caffe build is provided in the Faster R-CNN repository, interaction with the framework is done at every step of the way during the training procedure. Caffe is provided with online documentation [1]. The documentation contains installation instructions, DIY Deep learning tutorials, Application Programming Interface (API) documentation and several other resources. The API documentation gives a great insight into the aspects surrounding any Caffe-based implementation. The class list is well annotated, with mathematical formulations, descriptions and even use-cases. In this thesis, the most

```
I0507 01:56:30.953641 10476 net.cpp:380] Input 0 -> data
I0507 01:56:30.953641 10476 net.cpp:380] Input 1 -> labels
I0507 01:56:30.953641 10476 net.cpp:380] Input 2 -> labels_weights
I0507 01:56:30.953641 10476 net.cpp:380] Input 3 -> bbox_targets
I0507 01:56:30.953641 10476 net.cpp:380] Input 4 -> bbox_loss_weights
I0507 01:56:30.953641 10476 layer_factory.hpp:74] Creating layer conv1_1
I0507 01:56:30.953641 10476 net.cpp:90] Creating Layer conv1_1
I0507 01:56:30.953641 10476 net.cpp:420] conv1_1 <- data
I0507 01:56:30.953641 10476 net.cpp:378] conv1_1 -> conv1_1
I0507 01:56:30.953641 10476 net.cpp:120] Setting up conv1_1
I0507 01:56:30.953641 10476 net.cpp:127] Top shape: 1 64 224 224 (3211264)
```

Figure 6.2: Excerpt from the Caffe log

important aspect for Caffe is in initializing training and its logging feature for debugging. The logging feature can be enabled at any time before the Caffe mex is executed by adding

```
Caffe.init_log(fullfile(pwd, 'Caffe_log'));
```

to provide user with the ability to assess the execution after the mex is terminated. The Caffe mex is terminated either via clearing the solution (finishing the training/testing), or via errors terminating the procedure. This is undoubtedly the most important tool to anyone starting off with training Faster R-CNN or any other MATLAB Caffe based NN solution. The Caffe debugging log will provide information if dimensionality throughout the CNN is wrong, the GPU is out of memory, or the input data is wrongly parsed. In addition to this, the Caffe debug log gives a detailed run-down of how Caffe handles each layer of the network, the hierarchy of all inputs and outputs through the network, and the required memory to store model-parameters. This can be seen in Figure 6.2. The last line in Figure 6.2 shows the dimensionality $[N \times \mathbf{K} \times \mathbf{W_1} \times \mathbf{H_1}]$ as well as the total storage required for the input and kernels in parenthesis.

6.3.2 Rebuilding Caffe

An important aspect to the Faster R-CNN MATLAB implementation is the use of CUDA for parallel computing. The implementation presented here is built on CUDA 6.5 SDK using Visual Studio 2013. Furthermore, the graphic units used in Faster R-CNN are of the Nvidia Maxwell and Tesla architecture (Titan, Titan Black, Titan X, K20, K40, K80). The implementation done in this thesis is on a newer Nvidia GTX 1080 which is built on the Nvidia Pascal architecture. This is not supported by the older CUDA 6.5, but rather by the newer CUDA 8.0. Any attempt at running Faster R-CNN in GPU-mode will result in errors, one in particular to note is the following:

```
Invalid MEX-file '.../matlab/Caffe_faster_rcnn/Caffe_.mexw64'
```

The general solution is rebuilding 'Caffe for Faster R-CNN' to create a mex that is supported by whatever software or hardware requirements the user may have. There is a official repository containing the necessary information and requirements to do this. It can be found at https://github.com/ShaoqingRen/Caffe/tree/faster-R-CNN. Going into the details surrounding the building of a new mex this way is of little use. The issues each user

might face will largely depend on their specific hardware, pre-installed libraries, OpenCV, Boost, Visual Studio version and several more factors.

6.3.3 Testing trained models in Faster R-CNN

Any user downloading the content of the Faster R-CNN repository gets access to a ZF-net model and a VGG-16 model trained on the VOC0712 data set. This is usually the first insight new users get to the implementation aspect of this framework. This is certainly fairly straight forward for anyone to use, as users without the proper Caffe build can run this in CPU mode with a simple change to the parameter:

```
opts.use_gpu = false;
```

However, as is covered earlier in this chapter, it is not viable to train a new model on a CPU.

6.3.4 Annotations

One of the challenges in this thesis has been reformatting annotations and modifying Faster R-CNN to accommodate, and correctly parse annotation-files from various sources. Faster R-CNN takes mark-up language annotations (XML) as input and parses every character to a single string. Parsing text this way requires precise delimiting. The wrong amount of whitespace will corrupt the parsing procedure. Faster R-CNN only parses annotations correctly if they follow the VOC standard annotation. An example has been provided in Figure 6.3**A**. Imagenet annotations follow the VOC annotation standard, however, they do not serve the correct identification for this training task as they come with the synset folder-tag and name-tag. This is seen in Figure 6.3**B**. It is not advised using MATLAB for editing or creating XML files, as the way MATLAB writes annotation files will lead to corruption, adding incorrect whitespace and additional heading tags. It is possible however, with some modifications to the Faster R-CNN framework.

6.3.5 Training

Having covered what is initially required to begin training Faster R-CNN on any data set it is time to look at the different aspects of the training procedure. It is advised to have a dedicated computer for the training purpose. At the very least, a dedicated graphics unit should be used. Variations in memory load from other applications can crash the training procedure at any time, as memory in the graphics unit is not pre-allocated, only pre-calculated. The current version (and most likely final version) of the MATLAB Faster R-CNN framework supports training on ZF-Net and VGG-16. The model structures, their learning-rates and amount of training iterations are described by the .PROTOTXT files as discussed in Section 6.2.1.

One of the challenges in employing Caffe for Faster R-CNN is its 4-stage training procedure. As discussed in the documentation paragraph, Caffe is executed for one single training procedure. This means that the 4-stage training procedure is accomplished by

```
<annotation>
      <folder>VOC2007</folder>
      <filename>000001.jpg</filename>
      <source>
            <database>The VOC2007 Database</database>
            <annotation>PASCAL VOC2007</annotation>
            <image>flickr</image>
            <flickrid>341012865</flickrid>
      </source>
      <owner>
            <flickrid>Fried Camels</flickrid>
            <name>Jinky the Fruit Bat</name>
      </owner>
      <size>
            <width>353</width>
            <height>500</height>
            <depth>3</depth>
      </size>
      <segmented>0</segmented>
      <object>
            <name>dog</name>
            <pose>Left</pose>
            <truncated>1</truncated>
            <difficult>0</difficult>
            <bndbox>
                  <xmin>48</xmin>
                  <ymin>240</ymin>
                  <xmax>195</xmax>
                  <ymax>371</ymax>
            </bndbox>
      </object>
</annotation>
```

```
<annotation>
      <folder>n03673027</folder>
      <filename>n03673027_36806</filename>
      <source>
            <database>ImageNet database</database>
      </source>
      <size>
            <width>500</width>
            <height>490</height>
            <depth>3</depth>
      </size>
      <segmented>0</segmented>
      <object>
            <name>n03673027</name>
            <pose>Unspecified</pose>
            <truncated>0</truncated>
            <difficult>0</difficult>
            <bndbox>
                  <xmin>1</xmin>
                  <ymin>183</ymin>
                  <xmax>484</xmax>
                  <ymax>454</ymax>
            </bndbox>
      </object>
</annotation>
```

A	B

A VOC annotation example. **B** Imagenet annotation example.

Figure 6.3: Example of a VOC and Imagenet XML annotation

solving 4 distinct training-solutions. Each solution is done as its own training step, with a single execution of the Caffe mex, with its own solver. This has an inherent disadvantage. If an error is done in the solver or input-parsing at a late training-stage, or the GPU lacks memory for the specific task, this will not be known to the user before hours of training has gone by. If training is done over night, this will waste a large amount of training time. The biggest challenge concerning this has certainly been memory control. Some training procedures require splitting up the data set into smaller chunks to be able to cache all the required images and their associated data in GPU memory. This in itself is a time consuming task, as all annotations, image-sets, and images have to be correctly split up to ensure that no errors are induced into the next pass of the training procedure. Due to this, on average, even with the knowledge obtained throughout the implementation, a single training procedure may take about 2-4 passes before the final model is obtained, each taking everything from 9 to 17 hours.

There is an aspect of debugging in parallel computing worth considering. The parallel computing interface in Faster R-CNN is limited to calling Caffe classes, which in turn employs CUDA functions. Any error in the parallel computing process is not returned to neither MATLAB or the Caffe debug logger. Crashes in CUDA functions are therefore very hard to pinpoint. The times these errors were encountered during the implementation for this thesis, were when annotations and images did not coincide, e.g., if image sizes did not correctly correspond to what is described in the annotations, or the annotations had been badly parsed due to the XML-structuring. These errors may not be completely obvious, but careful investigation of the image database cache (imdb cache) will surface such errors. Due to the errors encountered here, a variety of verification scripts were written

to keep tally of any badly formatted data sets. These errors were especially prominent in the imagenet data sets, which often had annotated bounding-boxes spanning outside the image frames, or were converted TIFF images, which Faster R-CNN is not able to process. Some imagenet synsets, originally meant for training, had to be completely removed due to corrupt annotations.

Chapter 7

Experiments

This chapter describes the different experiments conducted to evaluate CNN models within Faster R-CNN for several different data sets and training-regimes.

- Section 7.1 present the four experiments conducted in this thesis.
- Section 7.2 presents an evaluation of all experiments done on the custom test-set in order to evaluate the robustness of the detectors for the ASV application.

7.1 Experiments

Having covered the various data sets that are going to be used and the theory and implementation concerning Faster R-CNN, it is time to look into the experiments that is conducted in this thesis. Training a CNN is largely a heuristic and problem dependent task. It is often necessary to complete a large amount of time-consuming training-regimes to determine an the best performing CNN model for the task. This thesis does not modify any of the hyper-parameters for the CNN-models presented earlier, as it is beyond the scope of the thesis. What is presented here, is training-regimes done on the ZF and VGG-16 for different combinations of VOC2007, VOC2012, imagenet and the custom data set collected by the Autosea project and annotated by the author. This is done in pursuit of an improved detection of the boat class, as it is presented in Faster R-CNN [25].

7.1.1 Experiment 1: Recreating the ZF VOC0712 model

From the Faster R-CNN repository one can download the demo models for both VGG-16 and ZF-net fine-tuned on the different VOC data sets. These can be executed and used ”off the shelf”. However, there are a lot variables in the implementation procedure, in particular, the CUDA and Visual Studio version. The versions used in this thesis do not correspond to the one used in the Faster R-CNN repository when training the models. If there is any change to the classes in CUDA for any reason, this may have implications on

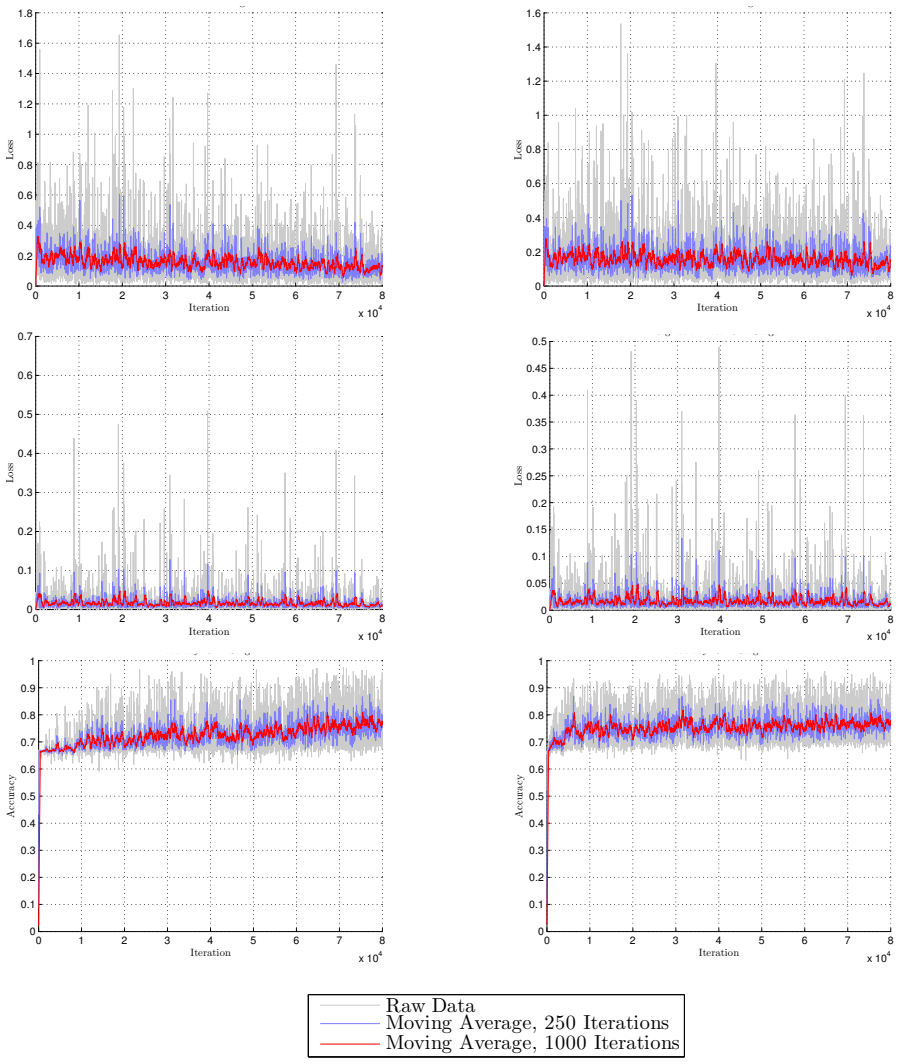
the model behavior during training. This experiment aims to validate the results obtained in Faster R-CNN [25]. This is achieved by fine-tuning a ZF-model from the pre-trained imagenet model on the VOC0712 data set, while validating on the VOC2007 test-set.

	Average Precision (%)			Average Precision (%)	
	Stage 1	**Stage 2**		**Stage 1**	**Stage 2**
Aeroplane	66	67	**Diningtable**	61	61
Bicycle	68	70	**Dog**	70	73
Bird	56	60	**Horse**	75	80
Boat	48	46	**Motorbike**	70	70
Bottle	33	33	**Person**	65	65
Bus	70	69	**Pottedplant**	30	31
Car	74	74	**Sheep**	63	64
Cat	76	78	**Sofa**	57	60
Chair	36	37	**Train**	72	71
Cow	65	70	**Tvmonitor**	62	62
			mAP	61	62

Table 7.1: Fast R-CNN model stage AP, ZF-net, VOC0712 [29]

Faster R-CNN provides the results shown in Table 7.1 for the ZF-Net VOC0712 fine-tuning procedure, validated on the VOC2007 test-set. These are not published in the Faster R-CNN article, rather, they are provided in the repository [29]. This table will serve as a benchmark for the AP surrounding the different classes. A correct replication of these results are not expected, as the convergence of each class depends on stochastic gradient descent for thousands of iterations. The ”Main Results” section of the repository shows a note concerning the random variation that occurs during different training-runs. Here, 5 independent trainings of ZF-net are shown to have a mAP 59.9% with a standard deviation of 0.39%. The note concerns training purely on the VOC2007 training-set. This is evident from a statement in the repository that the mAP of 59.9% is the result published in the Faster R-CNN paper. Table 2 in the Faster R-CNN paper shows that a mAP of 59.9% was achieved on the VOC2007 data set [25], but do not display similar results for the VOC0712 training-set. What is evident from Table 7.1 is that one instance of training on VOC0712 gives a 2.1% increase over the published results.

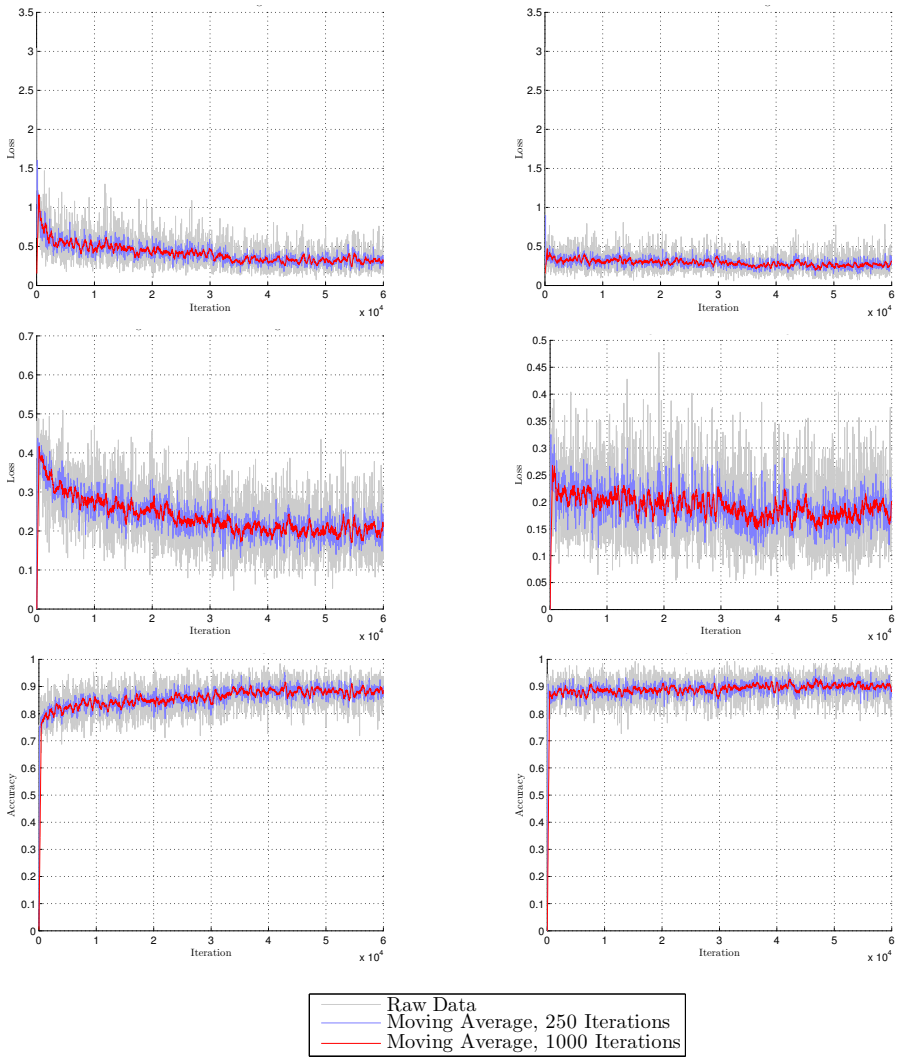
Figure 7.1 and Figure 7.2 shows the loss-functions and accuracy-evolution for the aforementioned training regime. The loss functions are as described in Equation (4.4) and Equation (4.6). These figures will serve as means of evaluating the convergence of the model that is trained in the experiment. The details of the per-iteration loss are dimly presented in grey for the reader. The trend between the different trainings however, is expected to be seen in the filtered moving average that is presented in blue and red. For comparison between the model trained in the experiments and the model provided in the repository, only a moving average will be shown.



A	**D**
B	**E**
C	**F**

First Row: Classification Loss, Middle Row: Regression Loss, Bottom Row: Accuracy.
A-C Stage 1 RPN
D-F Stage 2 RPN

Figure 7.1: RPN training loss and accuracy, ZF-Net, VOC0712, from [29].



A	**D**
B	**E**
C	**F**

First Row: Classification Loss, Middle Row: Regression Loss, Bottom Row: Accuracy.
A-C Stage 1 Fast R-CNN
D-F Stage 2 Fast R-CNN

Figure 7.2: Fast R-CNN training loss and accuracy, ZF-Net, VOC0712, from [29].

7.1.2 Experiment 2: Padding with Imagenet

In pursuit of increased AP for the boat class, this experiment pads the combined VOC0712 data sets with the imagenet data set discussed in Section 5.2. The training regime is the same as for Experiment 1, with Faster R-CNN running the ZF-net model. Keep in mind that the only ground-truth label and bounding-boxes in the imagenet images are of the boat class. Any noise from other classes, positively predicted by the model, will be counted towards a false positive for that class in the training-phase for these images. In creating the data set, the imagenet images are, at random, split into one training-set and one validation-set. Just as for the VOC data sets, these will approximately be split 50/50, with a small majority towards the validation set. This is to most closely relate the data set to the VOC data. None of the imagenet images will be used for testing purposes.

Data set	train	val	gt-boxes
VOC2007	2501	2510	12608
VOC2012	5717	5823	27450
Imagenet	794	865	1893

Table 7.2: Data sets split between training and validation data.

Table 7.2 shows how the different data sets are split between training data and validation data. The ground truth boxes for the total data set is shown in the last column. Padding the VOC0712 data set with the Imagenet data set increases the total amount of images with $\sim 10\%$. What is of larger interest here is the representation of the boat-class compared to the other 19 classes.

Data set	train	val	gt-boxes
VOC2007	81	100	290
VOC2012	260	248	850
Imagenet	794	865	1893

Table 7.3: Boat class representation between data sets

With the addition of the imagenet data set, the boat class has a $\sim 281\%$ increased data-representation compared to the VOC0712 data set. This can be seen in Table 7.3, where the amount of boat-images and ground-truth boxes are presented. The experiment will be evaluated on the VOC2007 test-set, which will give a quantifiable comparison to experiment 1.

7.1.3 Experiment 3: Single class training

Experiment 1 and 2 show the training regime of Faster R-CNN performed on all 20 VOC classes. Experiment 3 deviates from this, training the model only on the boat-class. The main idea here, is that the model should converge for real, positive, ground-truth cases of the boat class. In theory, it should give better model-parameters for such a case, if the training-data are sufficient. Reducing the data set, also means reducing the amount of negative examples for this class. In fact, all examples have positive ground-truth bounding boxes in them. However, the RPN network still has to localize the correct bounding-box, given an image. Here, an assumption is being made: There is such a vast amount of negative area in the images, and the RPN network does binary classification and bounding box regression on this negative area in addition to the positive. On these grounds, a smaller data set should still be sufficient for converging the model.

The data sets used for this experiment are the VOC0712+Imagenet data sets as described in Experiment 2. Training only on the boat class reduces the total data set to what is shown in Table 7.3. The experiment is evaluated on a subset of the VOC2007 test-set, which all contain ships in them. Although this is not officially disclosed in the VOC documentation [6], there are a total of 172 images in the VOC2007 test-set containing the class. These annotations are then reduced to only accommodate the boat-class, leaving any other class from the VOC class-list in the images as background. This experiment will also evaluate another assumption. Given fewer classes at the output of the CNN, less iterations of stochastic gradient descent should be necessary to obtain convergence. In fact, too many iterations may induce overfitting on the data set.

Experiment	CNN-Model	RPN Iterations		Fast R-CNN Iterations	
		$\eta = 0.001$	$\eta = 0.0001$	$\eta = 0.001$	$\eta = 0.0001$
E_{3A}	ZF-Net	60k	20k	30k	30k
E_{3B}	VGG-16	60k	20k	30k	30k
E_{3C}	VGG-16	60k	20k	30k	10k

Table 7.4: E_3: Definition of sub-experiments

Table 7.4 shows the different experiments conducted on the single class data set. Here, Experiment E_{3A} is the original amount of iterations as can be seen in Figure 4.5 for the ZF-Net model. Experiment E_{3B} has the same amount of iterations on the VGG-16 model, while E_{3C} decrease the amount of Fast R-CNN iterations to allow for the investigation of the above-mentioned assumption. The assumption will be confirmed if a significant increase in AP is observed for E_{3C} over E_{3B}.

7.1.4 Experiment 4: Padding with the Custom training-set

Currently, the experiments presented in this chapter have been limited to images made and annotated, for the purpose of creating a data set for image classification in the general case. This experiment investigates the possibilities of creating an object detection network, trained on images from the area that the vessel is to commute. To accomplish this, a

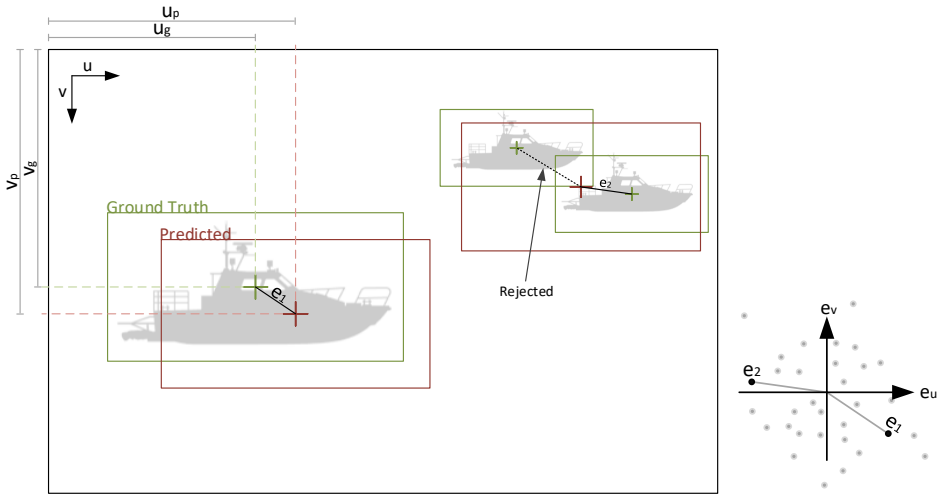
baseline data set is still needed. Padding the data sets shown in Experiment 2 with the custom data set allows for general convergence for the boat-class, and allows us to investigate the hypothesis of Section 5.3 that data provided from a pre-defined path diminishes the amount of false positives predicted on the environment in the image. This gives the total data set VOC0712+Imagenet+Custom. This experiment is trained using the default iterations described in Faster R-CNN and in Experiment E_{3A}. The experiment is conducted on the VGG-16 model with the binary training regime on the boat class.

7.2 Evaluation on the Custom test-set

So far, Experiment 1-4 have been evaluated on the VOC2007 test-set or a subset of this. While this certainly gives an indication of the robustness of the different trained models, it is not necessarily representative of the performance for the ASV application. To evaluate this, all models are tested on the same set of images. Their mAP will be computed on the custom test-set described in Section 5.3. This evaluation also provides useful information concerning testing a CNN on data closely related to the training data. Experiment 4 has the inherent advantage of being related to this test-set via the custom training-set. This will evaluate the hypothesis of Section 5.3, that training-data collected on a predefined path eliminates more false positives in the model.

An addition to this evaluation is the introduction of a metric to determine detection-offset to ground-truth. Where the precision/recall-curve provides valuable information on robustness for every detection with IoU > 0.7, the proposed metric will provide information on the offset for all true-positive detections in every image of the custom test-set. The metric is defined as follows: We define the center of a ground-truth bounding-box to be given by the coordinates (u_g, v_g) and the predicted bounding-box (u_p, v_p) in the image plane spanning (n_u, n_v) pixels. The error between these two points in the image plane is given by $(u_e, v_e) = (u_g - u_p, v_g - v_p)$. The resulting euclidean distance is $e = \sqrt{u_e^2 + v_e^2}$. For each positive detection, store the error with the shortest euclidean distance to a ground-truth box.

This method allows for multiple detections of a single ground-truth object. It is a metric that considers the measurement noise that can be expected using this detection method. This should serve as valuable information for initial gating when modelling measurement noise in the tracking-problem, employing extended kalman filtering or similar techniques.



Boat silhouette Courtesy: https://clipartfest.com

A	B

A Illustration showing prediction's center related to ground truth. **B** Illustration of the error-plane, mapping all distances for all true positive predictions.

Figure 7.3: Illustration of euclidean prediction error.

Part IV

Results and Discussion

Chapter 8

Results and Discussion

This chapter presents the results and discussion for the experiments described in chapter 7. The results and discussion for each experiment is presented independently, referring only to preceding experiments.

- Section 8.1 presents the results for experiment 1 described in Section 7.1.1.
- Section 8.2 presents the results for experiment 2 described in Section 7.1.2.
- Section 8.3 presents the results for experiment 3 described in Section 7.1.3.
- Section 8.4 presents the results for experiment 4 described in Section 7.1.4.
- Section 8.5 presents the results for the custom evaluation described in Section 7.2.
- Section 8.6 presents a video of the two best performing detectors, and analyses their performance.
- Section 8.7 summarizes the results obtain in the various experiments.

8.1 Experiment 1: Recreating the ZF VOC0712 model

This experiment aims to validate the results published in Faster R-CNN for the ZF-net model on the VOC0712 data set. This is done as a preliminary step towards padding the data set to increase the accuracy in later experiments.

Notation

The results for the experiment is denoted E_1 while the results provided in the repository for Faster R-CNN will be denoted F_{ZF} to distinguish this from the experiments conducted in the thesis.

Results

Table 8.1 shows the results obtained during a single run of training Faster R-CNN on VOC0712. The results presented are for the classification stages (Fast R-CNN) according to the 4-stage alternate training regime in Figure 4.5. The mAP obtained is identical for both stages of the experiment and the provided model. Included in these results are the precision and recall curve for both stages of Fast R-CNN, shown in Figure 8.1. This is computed according to Equation (3.15). Figure 8.2 compares the loss and accuracy for the RPN training to what is published in the Faster R-CNN repository and Figure 8.3 shows the loss and accuracy for the Fast R-CNN training. Both these figures display the 1000 iteration moving average, and do not concern the raw data as in Figure 7.1 and Figure 7.2.

	Average Precision (%)					Average Precision (%)			
	F_{ZF}	E_1	F_{ZF}	E_1		F_{ZF}	E_1	F_{ZF}	E_1
	Stage 1		**Stage 2**			**Stage 1**		**Stage 2**	
Aeroplane	66	66	67	68	**Diningtable**	61	58	61	62
Bicycle	68	69	70	70	**Dog**	70	69	73	71
Bird	56	57	60	60	**Horse**	75	78	80	77
Boat	48	50	46	49	**Motorbike**	70	68	70	71
Bottle	33	32	33	33	**Person**	65	65	65	66
Bus	70	71	69	70	**Pottedplant**	30	30	31	30
Car	74	73	74	75	**Sheep**	63	63	64	62
Cat	76	76	78	76	**Sofa**	57	56	60	60
Chair	36	37	37	38	**Train**	72	71	71	69
Cow	65	66	70	68	**Tvmonitor**	62	59	62	59
					mAP	61	61	62	62

Table 8.1: E_1: Fast R-CNN model stage AP, ZF-net. Comparison between results obtained in Faster R-CNN repository [29] and the recreation done on the VOC0712 training data.

Discussion

There is a clear similarity in the mAP between the model trained in E_1 and the one provided in the Faster R-CNN repository F_{ZF}. Some slight variations, such as the increased AP for the boat class, and decreased AP for the tvmonitor class is expected using SGD optimization. From Figure 8.2 and Figure 8.3 it is certainly possible to deduce that both models converge at the same rate, with the small exception being the classification loss in the second stage of Fast R-CNN in Figure 8.3**D**. Here, the model trained in this experiment shows a slightly superior convergence overall throughout the training. It comes as no surprise that these two models share such similar characteristics. However, validating this provides a verifiable benchmark for the subsequent experiments which adds and subtracts from the data sets applied here. There are a lot of variables in play when implementing such a large framework, utilizing several third party softwares. One cannot take for granted that all training and testing images are the same and all annotations un-revised, is certainly not good practice going into an experiment handling large amounts of data.

There are also other changes, such as software and hardware differences that could come into play, changing the outcome of the training procedure.

The deviation for the boat class between F_{ZF} and E_1 in Table 8.1 came as a surprise. Although this is most certainly a positive outcome, it is a result of the random nature of the SGD optimization. This in turn becomes evident if we regard the mAP of both the E_1 and F_{ZF} model. Both models present the exact same mAP across the board, for both stages of Fast R-CNN. Evaluating the RPN training procedure is unfortunately not as easy. The nature of the RPN boils down to its binary classification (object versus not object), as well as the box-regressor. In each image, these losses are calculated for a vast amount of bounding-boxes. The top 2000 regions are then computed using Non-Maximum Suppression. The output of the NMS is not stored in the current version of Faster R-CNN, making it hard to evaluate which boxes, and what classification score these boxes were chosen for.

Looking at the precision and recall for the two seperate Fast R-CNN stages in Figure 8.1 it can be seen a somewhat larger rate of high precision/low recall classifications for the boat-class in the first stage. The constituting factor for the lower AP for this class at the output stage can be due to several factors. It could certainly be attributed to the under-representation of the class in the data set. The more iterations of back-propagation, the more of the data set is used. The cause and effect of this, is a bias toward high representation in the data. This is certainly not the deciding factor, things like geometry and texture play a major roll in distinguishing the different classes. If one were to train a model on i.e. humans and boxes, the geometry and textures are more easily distinguishable, from the large scale of rounded and square edges, to the smaller scales of containing features like eyes, to not containing any features, or simple textures. In this case, we have boats, buses, and cars, which are not inherently similar at a large scale, but that share more and more features at lower scales such as windows. The bus class presents a argument against poor data representation being the cause of low AP here: It has low data representation in Table 8.1, but the AP is remarkably good. This in fact that in both the VOC2007 and the VOC2012 training data the bus class has about the same representation as the boat class. This is seen in Table 5.2 and 5.3. However, the bus class shares great similarities with the highly represented car class. This in turn, during optimization, only fine-tunes some of the weights and parameters that correctly predict the car class to distinguish the features of a bus to that of a car.

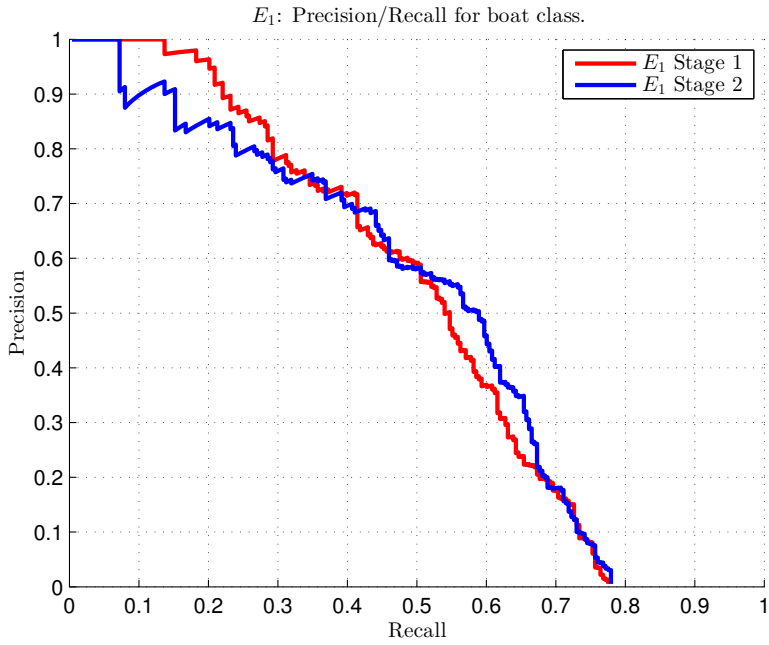
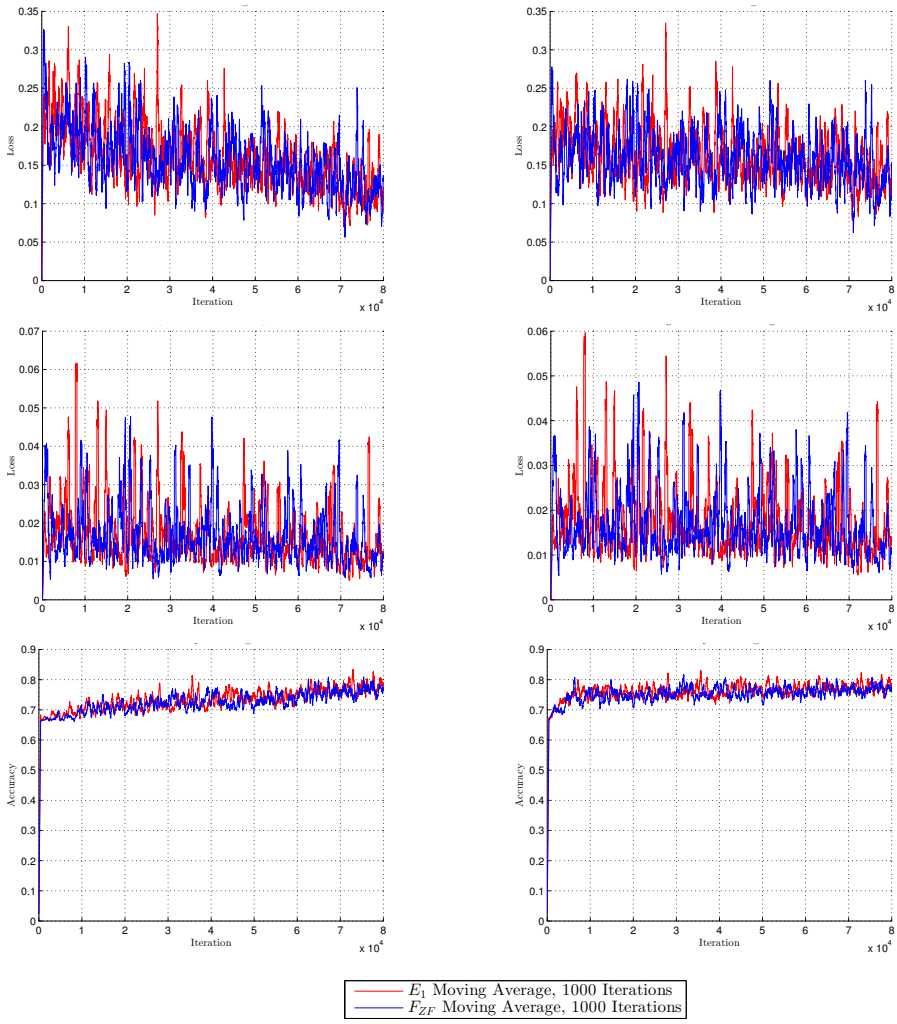


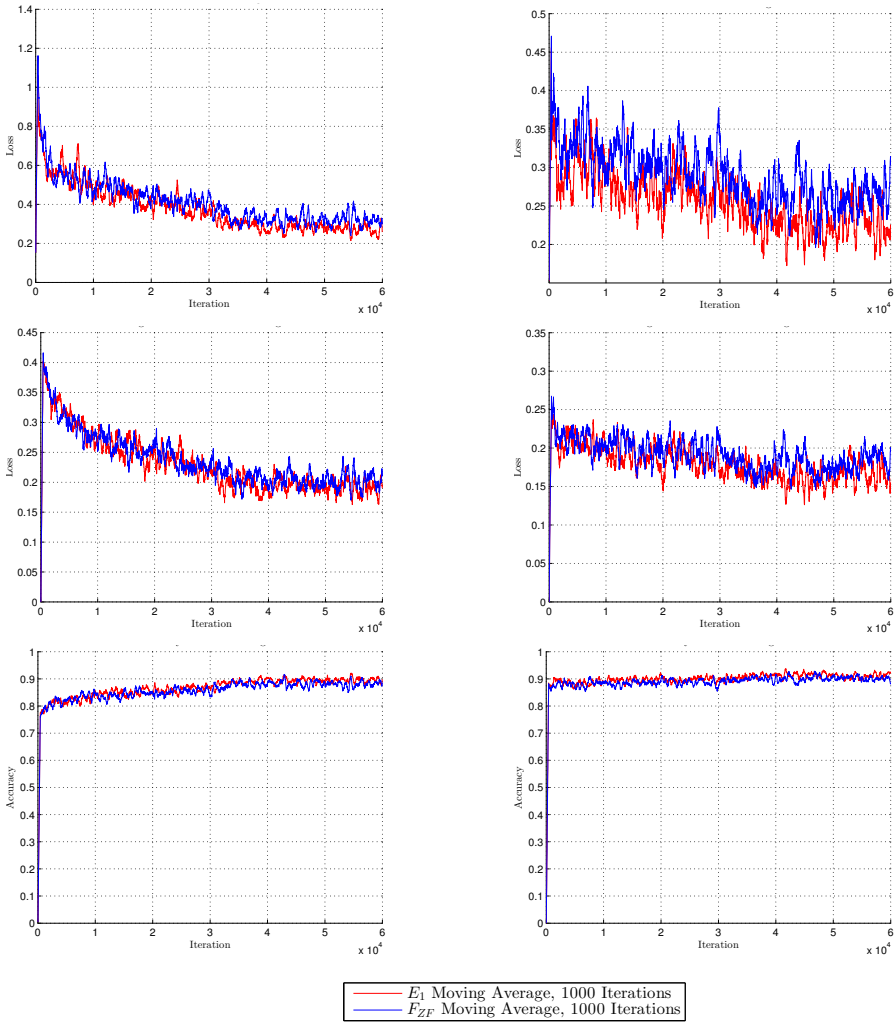
Figure 8.1: E_1: Precision vs. Recall for the boat class for the ZF-net model trained on VOC0712.



A	**D**
B	**E**
C	**F**

First Row: Classification Loss, Middle Row: Regression Loss, Bottom Row: Accuracy.
A-C Stage 1 RPN
D-F Stage 2 RPN

Figure 8.2: E_1: RPN Loss function and accuracy comparison between results obtained in Faster R-CNN repository [29] and the recreation done in E_1 for VOC0712 training data.



A	**D**
B	**E**
C	**F**

First Row: Classification Loss, Middle Row: Regression Loss, Bottom Row: Accuracy.
A-C Stage 1 Fast R-CNN
D-F Stage 2 Fast R-CNN

Figure 8.3: E_1: Fast R-CNN Loss function and accuracy comparison between results obtained in Faster R-CNN repository [29] and the recreation done in E_1 for VOC0712 training data.

8.2 Experiment 2: Padding with Imagenet

In pursuit of better detection of the boat class, the results presented in this experiment compares the training on the padded data set to the original results presented in E_1.

Notation

The results for this experiment are denoted E_2.

Results

Table 8.2 compares the results obtained with the padded data set (E_2), to that obtained in E_1 for all VOC classes. Figure 8.5 is an exhibit of two images showing the object detection performance for both the E_1 and E_2 model. These images are taken from the VOC2007 test-set, the data set under which both experiments are validated. The images were chosen as the two first in the VOC2007 test-set to contain the boat class. This can be verified through inspection of the VOC2007 devkit image-sets. Figure 8.4 compares the precision and recall for the boat class in E_1 and E_2.

	Average Precision (%)					Average Precision (%)			
	E_1	E_2	E_1	E_2		E_1	E_2	E_1	E_2
	Stage 1		**Stage 2**			**Stage 1**		**Stage 2**	
Aeroplane	66	63	68	66	**Diningtable**	58	58	62	59
Bicycle	69	68	70	69	**Dog**	69	70	71	72
Bird	57	56	60	60	**Horse**	78	75	77	75
Boat	50	51	49	52	**Motorbike**	68	70	71	72
Bottle	32	31	33	32	**Person**	65	64	66	65
Bus	71	69	70	72	**Pottedplant**	30	29	30	30
Car	73	73	75	74	**Sheep**	63	61	62	63
Cat	76	76	76	77	**Sofa**	56	57	60	59
Chair	37	35	38	37	**Train**	71	71	69	72
Cow	66	67	68	71	**Tvmonitor**	59	59	59	60
					mAP	61	61	62	62

Table 8.2: E_2: Fast R-CNN model stage AP, ZF-net. Comparison between the model trained on VOC0712 and the model trained on VOC0712+Imagenet.

Discussion

Although there is uncertainty related to this method, there is a marginal improvement for the E_2 model in Table 8.2. Compared to E_1 the boat-class has achieved 3% increased AP at the output of the last stage of training. Compared to F_{ZF} there is a 6% increased AP. The experiment also gives further indication that increased exposure to the class during training gives better performance. Even though this might certainly seem like an obvious assumption, increased exposure may also lead to overfitting, ultimately decreasing the AP.

It is also worth noting that this is the first experiment where the stage 2 AP for the boat class is higher than the first stage. Figure 8.4 shows the precision and recall for the training. In this experiment it is compared to what was achieved in E_1. Initially, after the first stage of Fast R-CNN training, the E_1 model achieves a higher rate of high precision/low recall detections. After the second stage, however, E_2 has retained the high-precision cases over a larger recall span. This is indicative of a slight performance increase in images containing multiple ground truth objects.

Also presented here are two images comparing the two experiments done so far shown in Figure 8.5. These were added to provide the reader a more intuitive understanding of what such a difference in AP constitutes. This difference is most notable in Figure 8.5**A**, where, compared to E_1, E_2 correctly detects three additional vessels. It is worth noting that the rightmost detected vessel in this figure is rather poorly classified, with only a 63% certainty. Figure 8.5**B** gives a larger, more detailed perception of a ship for the models to detect and classify. The notable difference in this case is the bounding-box size for both detections. Although marginal, the E_1 model is certainly closer to what a human would draw for a bounding-box, leaving the centroid of the bounding-box closer to ground-truth.

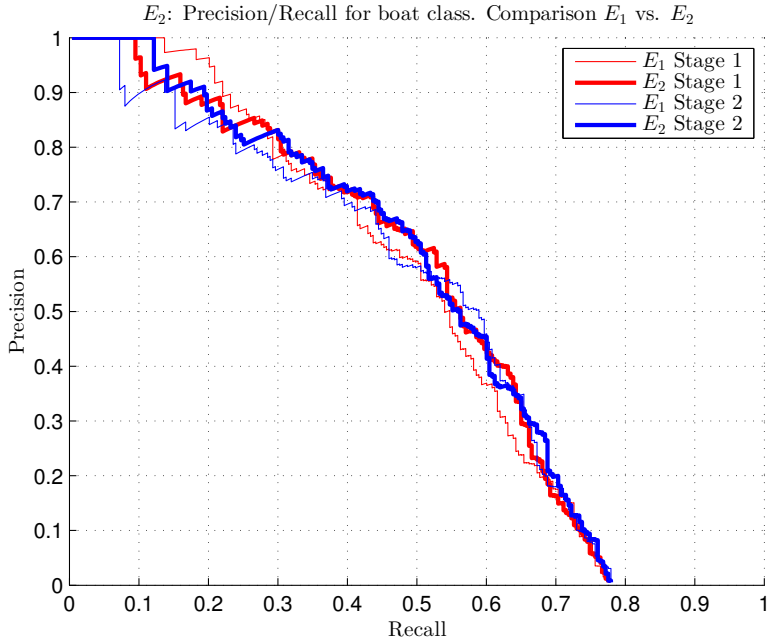


Figure 8.4: E_2: Precision vs. Recall for the boat class comparing the ZF-net model trained on VOC0712 and the ZF-net model trained on VOC0712+Imagenet.

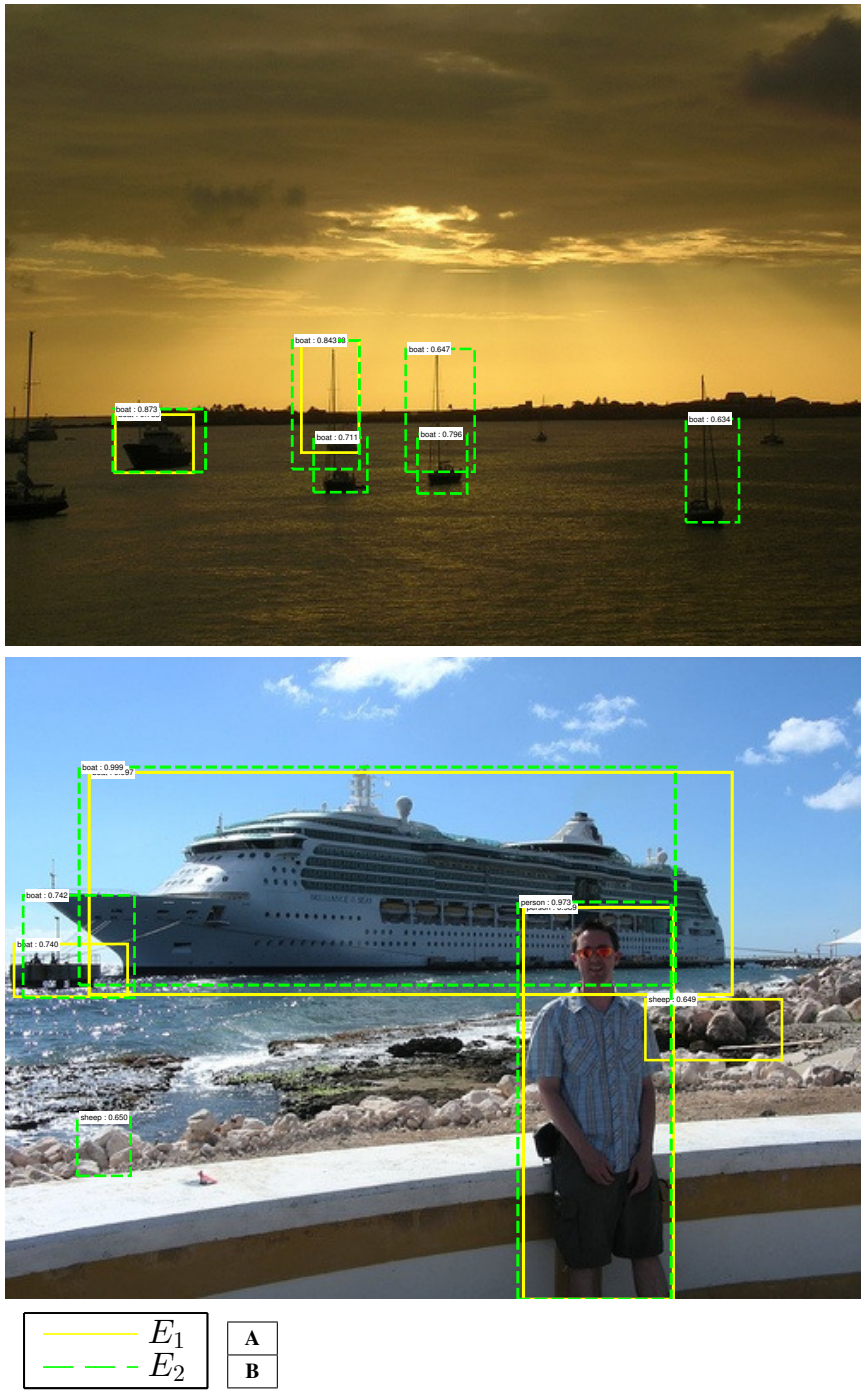


Figure 8.5: E_2: Output samples from the VOC2007 test-set for the ZF-net model trained on VOC0712 and the ZF-net model trained on VOC0712+Imagenet.

8.3 Experiment 3: Single class training

This section presents and evaluates the results obtained from training various models on a subset of the VOC and imagenet data sets containing the boat class.

Notation

The results for this experiment is denoted E_3. Sub-experiment E_{3A} is ZF-net with default iterations. E_{3B} is VGG-16 with default iterations. E_{3C} is VGG-16 with reduced Fast R-CNN fine-tuning.

Results

Table 8.3 displays the AP for the respective sub-experiments. Figure 8.6 compares the AP for all models evaluated on the VOC2007 test-set. Figure 8.7 compares the three different RPN training-regimes, with a 2000-iteration moving average for better visualization. Figure 8.8 compares the Fast R-CNN stages in the same regard. Lastly, Figure 8.9 revisits the exhibition in Figure 8.5 for a sample comparison between the three sub-experiments.

	AP (%)	
	Stage 1	**Stage 2**
E_{3A}	57	63
E_{3B}	66	67
E_{3C}	66	65

Table 8.3: E_3: Fast R-CNN model stage AP. Comparison between the default iteration ZF-net model, default iteration VGG-16 model and reduced iteration VGG-16 model.

Discussion

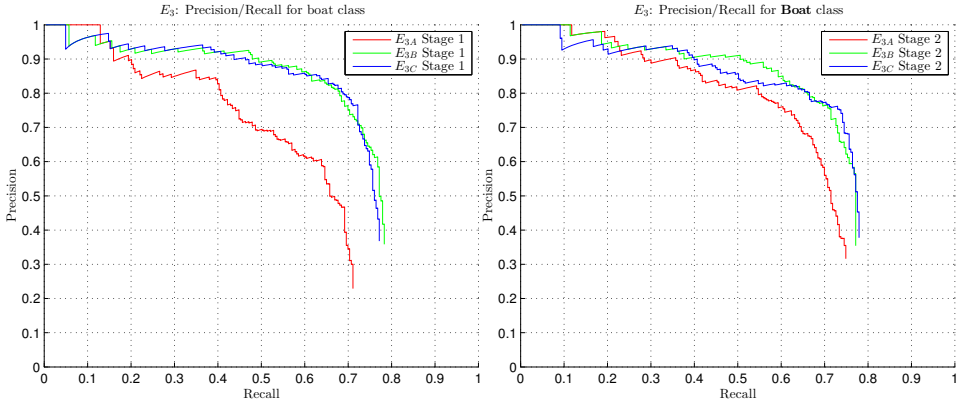
This experiment has investigated three different methods for object-detection using a binary classifier. The first experiment uses the ZF-net model with default iterations (E_{3A}). The second uses the deeper VGG-16 model (E_{3B}) and the last is VGG-16 with reduced iterations for the Fast R-CNN training stages (E_{3C}). This is the first experiment conducted on the VGG-16 model. As this experiment is evaluated on a subset of the VOC2007 test-set, it will not be compared to the 20-class models in Experiment E_2. For this experiment, VGG-16 certainly outperforms ZF-net in every regard, with a 4% superiority on the VOC2007 test-set at the output stage. The reduced VGG-16 model trained in Experiment E_{3C}, although more or less equivalent with E_{3B} after the first stage of training, loses robustness to the fine-tuning of the fully connected layers in stage 2.

Figure 8.6 shows the rather remarkable increase in AP for the ZF-model between the two stages. Although not conclusive in any regard, the ZF-net model may have benefited from a larger amount of iterations in stage 2. Although reduced iteration VGG-16 model has a decreased AP between the two stages, it does present some interesting results. 8.6**B**

shows this model having an increased recall at the lower precision range ($r >\sim 0.68$), indicating that, even though it does detect a larger amount of false positives, it also detects more ground-truth objects than the full iteration VGG-16 model in E_{3B}. This is largely overshadowed however by the great precision E_{3B} displays in the mid recall range ($\sim 0.4 < r <\sim 0.62$) over E_{3C}.

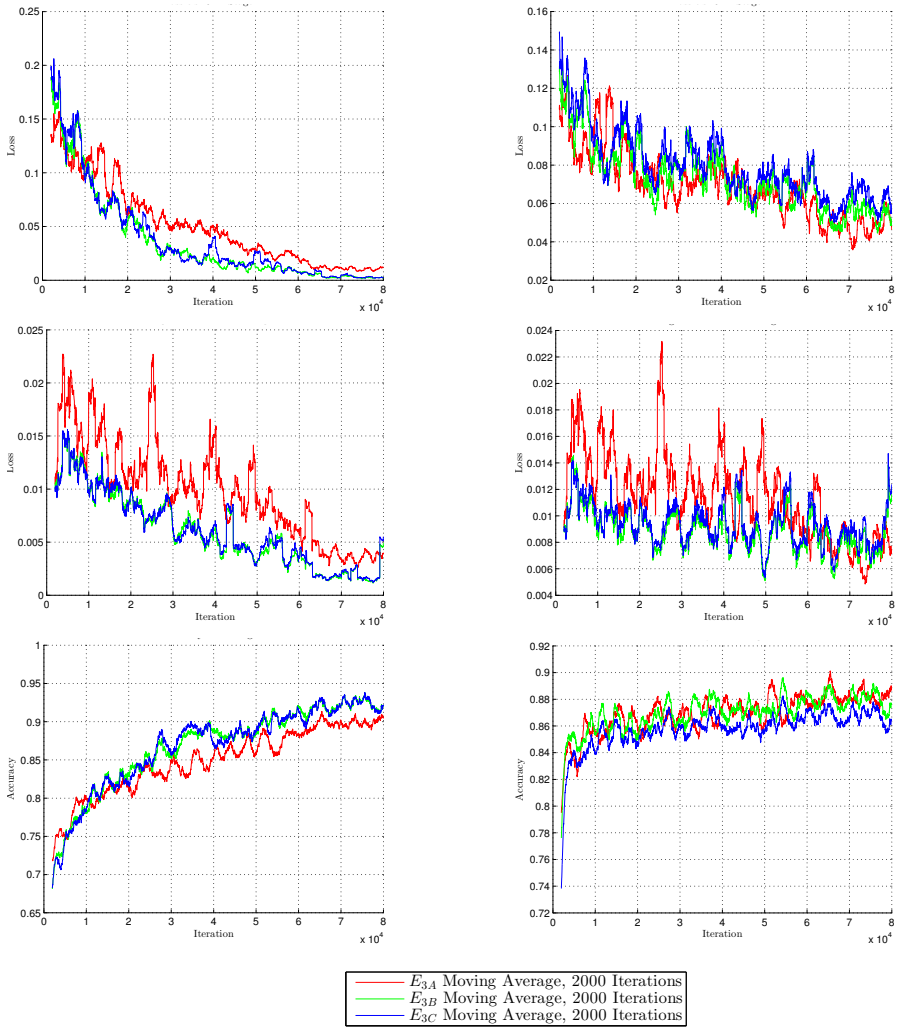
The training loss and accuracy is also presented for a more general ZF-net versus VGG-16 comparison. The general conclusion here is the same, VGG-16, both with reduced iterations and full iterations has a better convergence in most aspects of the training. There are some expections, such as the sudden increase in RPN regression loss in both stages for the VGG-16 models seen in Figure 8.7**B** and Figure 8.7**E**.

In Figure 8.9 we again revisit the sample images from E_2. Figure 8.9**A** display a remarkable difference of precision and recall, and give grounds for not comparing the AP in this experiment to the ones evaluated on the full VOC2007 test-set. From Table 8.3 we have an AP of 63% for ZF-net, which is a 9% increase from the 20-class VOC+imagenet ZF-net model in E_2 from Table 8.2. It is however clear that it does not perform nearly as good as the E_2 model on Figure 8.9**A**. Both E_{3B} and E_{3C} display good results, in Figure 8.9**A**, outperforming E_2 by detecting an additional ship (leftmost in figure), and containing all detections within a single bounding-box. Note the increased overall softmax classification-score for all detections in Figure 8.9 compared to Figure 8.5. This is due to the relative nature of the softmax-function as seen in Equation (3.5) and does not measure any quantifiable increase in robustness by itself, but should be considered when choosing a classification threshold by heuristics. A heuristic from the images displayed here and in the E_2 results is that accepting detections over 0.9 classification score using binary detection is approximately equivalent to a lower thresholding of the 20-class model at 0.65.



A **B** **A** Stage 1 Precision/Recall. **B** Stage 2 Precision/Recall.

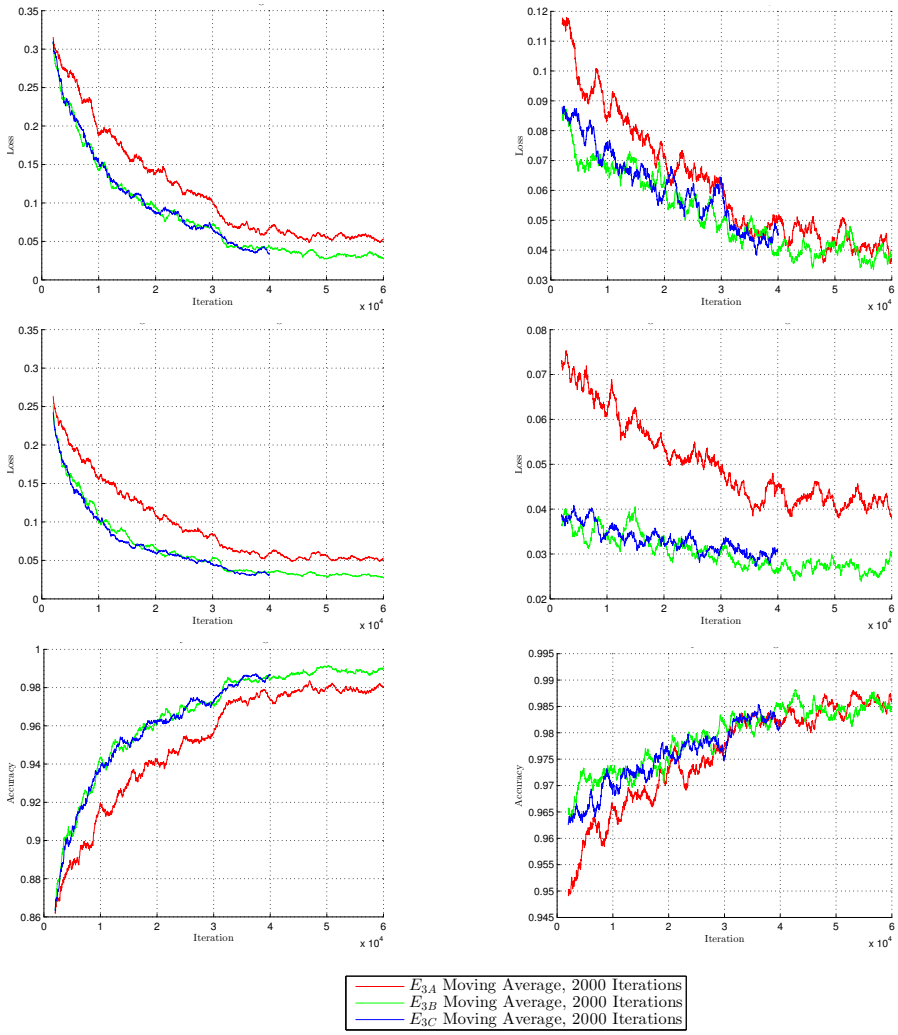
Figure 8.6: E_3: Precision/Recall comparison between the default iteration ZF-net model, default iteration VGG-16 model and reduced iteration VGG-16 model.



A	**D**
B	**E**
C	**F**

First Row: Classification Loss, Middle Row: Regression Loss, Bottom Row: Accuracy.
A-**C** Stage 1 RPN
D-**F** Stage 2 RPN

Figure 8.7: E_3: RPN Loss function and accuracy comparison, VOC0712+Imagenet training on the default iteration ZF-net model, default iteration VGG-16 model and reduced iteration VGG-16 model.



A	**D**
B	**E**
C	**F**

First Row: Classification Loss, Middle Row: Regression Loss, Bottom Row: Accuracy.
A-C Stage 1 Fast R-CNN
D-F Stage 2 Fast R-CNN

Figure 8.8: E_3: Fast R-CNN Loss function and accuracy comparison, VOC0712+Imagenet training on the default iteration ZF-net model, default iteration VGG-16 model and reduced iteration VGG-16 model.

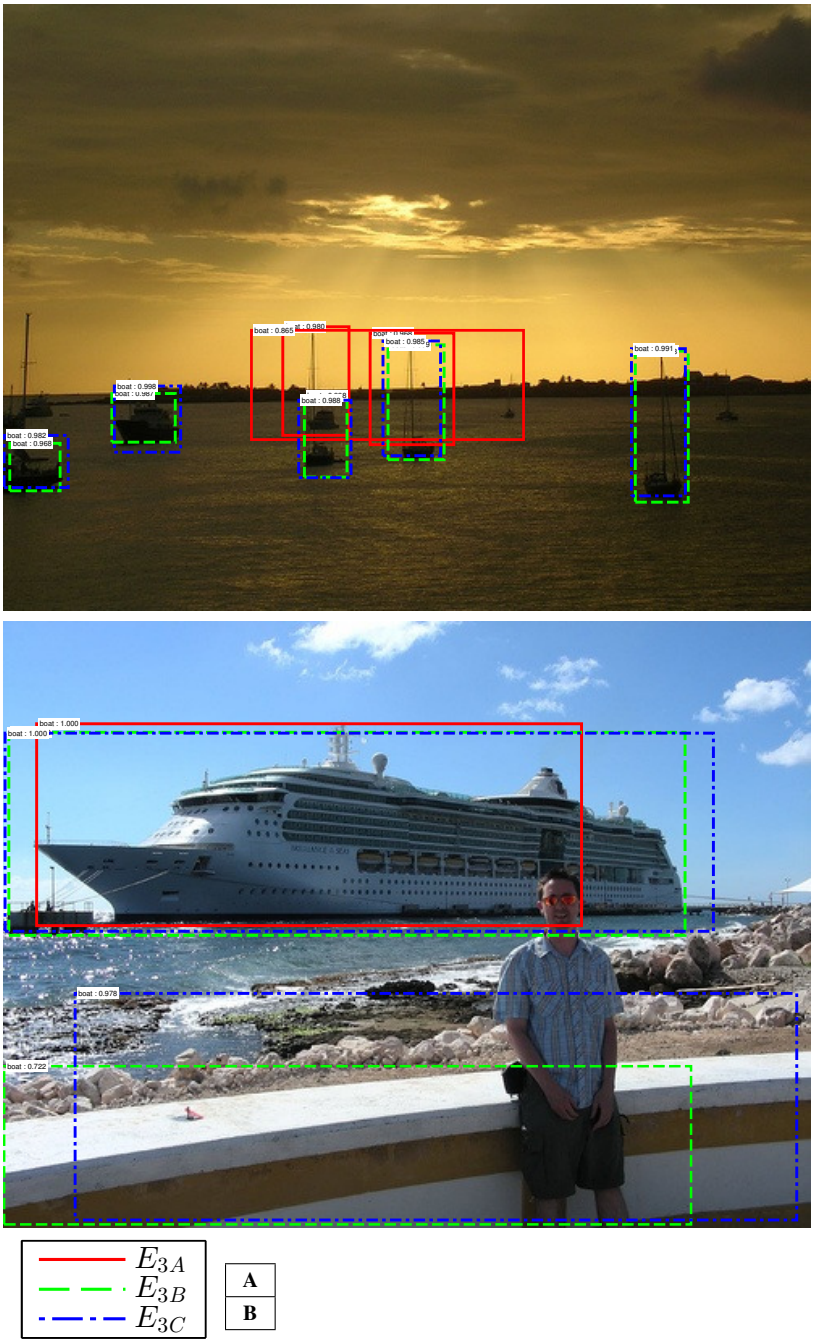


Figure 8.9: E_3: Output detection-samples from the VOC2007 test-set for the default iteration ZF-net model, default iteration VGG-16 model and reduced iteration VGG-16 model.

8.4 Experiment 4: Padding with the custom data set

This section presents and evaluates the results obtained from training the best performing model from the previous experiment (E_{3B}). The difference lies in padding the data set with the custom data set containing an additional 100 images of boats taken in along the Trondheim harbor and fjord.

Notation

The results for this experiment is denoted E_4. Comparison is done with the best performing model from the previous experiment with notation E_{3B}.

Results

Table 8.4 compares the results obtained from the model trained on a data set padded with 100 addition images with that obtained in E_{3B}. The results are from testing on the VOC2007 test-set.

	AP (%)	
	Stage 1	**Stage 2**
E_{3B}	66	67
E_4	67	67

Table 8.4: E_4: Fast R-CNN model stage AP. Comparison between the default iteration VGG-16 model trained on VOC0712+Imagenet and VGG-16 model trained on VOC0712+Imagenet+Custom.

Discussion

Table 8.4 display similar results between the two models. This is due to the fact that the custom data set do not have a large impact on the general training outcome. On a data set such as the VOC2007 test-set this will make little difference. The purpose of this experiment was not to increase the robustness generally, but rather investigate the fixed-path detector performance. This will be further evaluated in Section 8.5.

8.5 Evaluation on the Custom test-set

So far we have seen the general performance of several different trained Faster R-CNN models. They have been evaluated on a test-set meant as a general benchmark for researchers to use when competing in the Pascal VOC challenge [5]. This does not inherently constitute good performance for the ASV application. The results presented in this section are the outcome of testing all models, including the stock models from the Faster R-CNN repository [29] on the custom test-set.

Notation

$E_{\#\text{x}}$ refers to the experiments conducted in this thesis, with $\#$ being the experiment number, while x denotes possible sub-experiments.

Results

Table 8.5 displays the AP for each model subject to the custom test-set. The graph presented in Figure 8.10 displays the precision and recall curve for all experiments on this test-set. Figure 8.11 shows the euclidean distance error for the predicted true positive bounding-boxes to ground truth in the image plane. Lastly, in Figure 8.12 an exhibition of images is presented. These images show the detections for all models in this thesis on a single scene. Figure 8.13 show a variety of scenes for the two best performing models.

	AP (%)		tp (of 351 total)
	Stage 1	**Stage 2**	**Stage 2**
E_1	6	7	74
E_2	5	6	62
E_{3A}	11	11	61
E_{3B}	18	17	101
E_{3C}	17	17	93
E_4	52	55	240

Table 8.5: Model stage AP from the Custom test-set, all models.

Discussion

It is evident that this is a challenging task for all the models. Table 8.5 certainly give means to the hypothesis, with the E_4 model outperforming all the prior models. In this evaluation it also becomes clear that for the ASV task, the single-class models outperform the 20-class ones. The E_{3B} and E_{3C} experiments show a 10% increased AP over E_1.
The distance error in Figure 8.11 gives further grounds for the hypothesis, but should be analysed with some care. Some of the ships in the custom test-set do also appear in the custom training-set. This could certainly induce overfitting for the E_4 model. The results shown in 8.4 show that the E_4 model performs identically to the E_{3B} model on the VOC2007 test-set, which is evidence that overfitting on the new data is not the case. The custom training-set constitutes only 4% of the total training-data for E_4, so overfitting is unlikely. The two 20-class experiments (E_1 and E_2) display a larger amount of true positive detections over a wider range of softmax scores. This is due to the relative nature of the calculation in Equation (3.5). An interesting observation is the inherently wider range of true positive detections for these two models. True positive cases implies a IoU > 0.7. For the detections to be displaced by such a large amount, the positive predictions are of larger objects. Different models may also have positive predictions of different ground-truth objects, allowing for a degree of uncertainty to the method. Looking at the trends however, one can conclude that the single-class models generally perform better taking

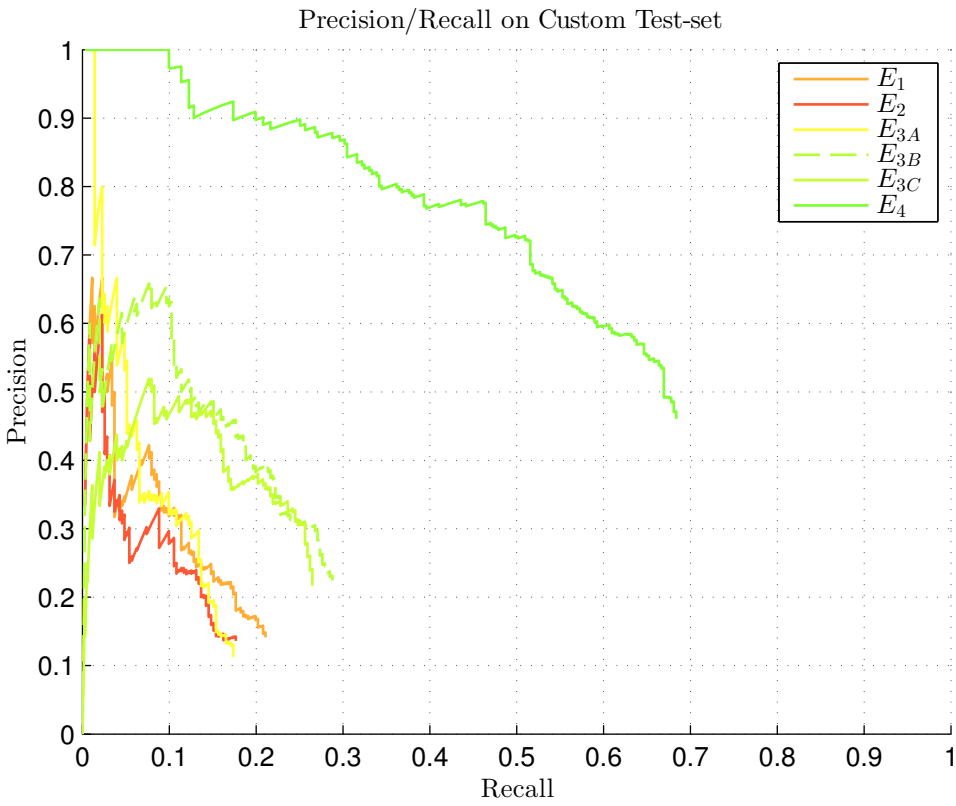
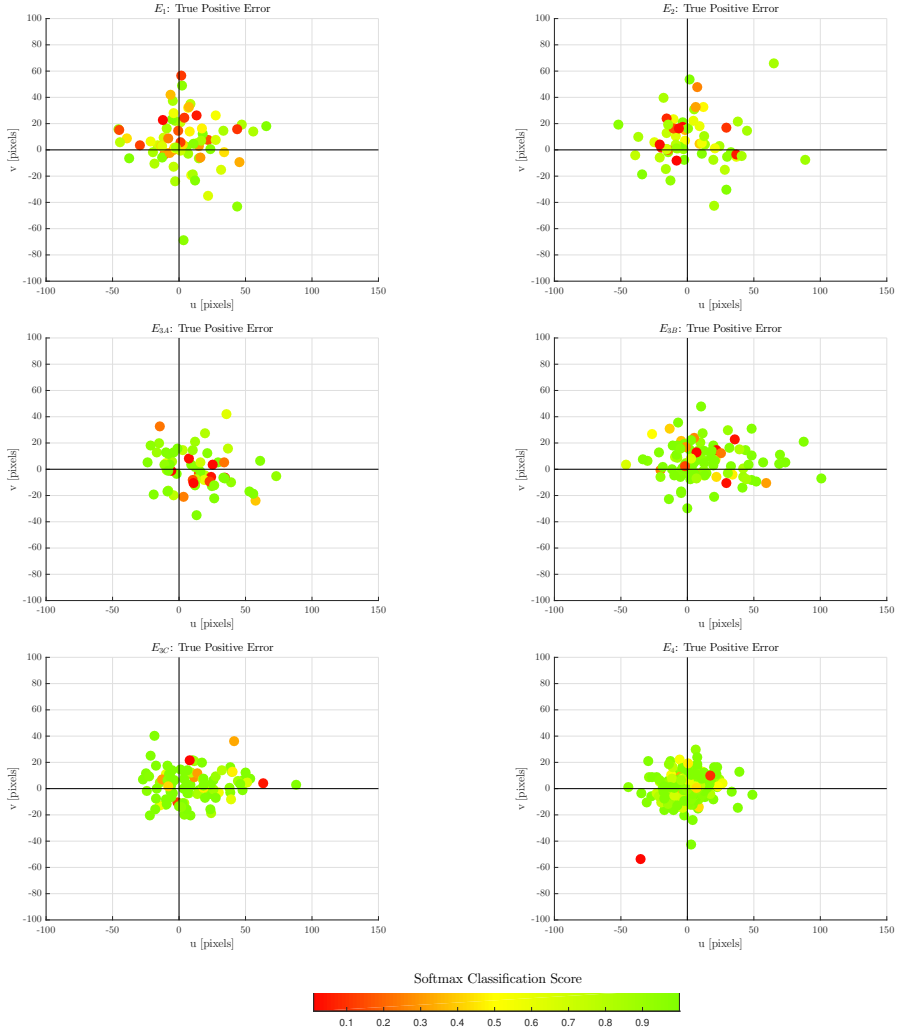


Figure 8.10: Precision/Recall for all models on the Custom test-set

the AP into account. All models exhibit some true positive detections that are at the low end of the classification score, indicating affiliation to the background, or other classes. This provides a heuristic for choosing a classification-threshold when using one of these models.

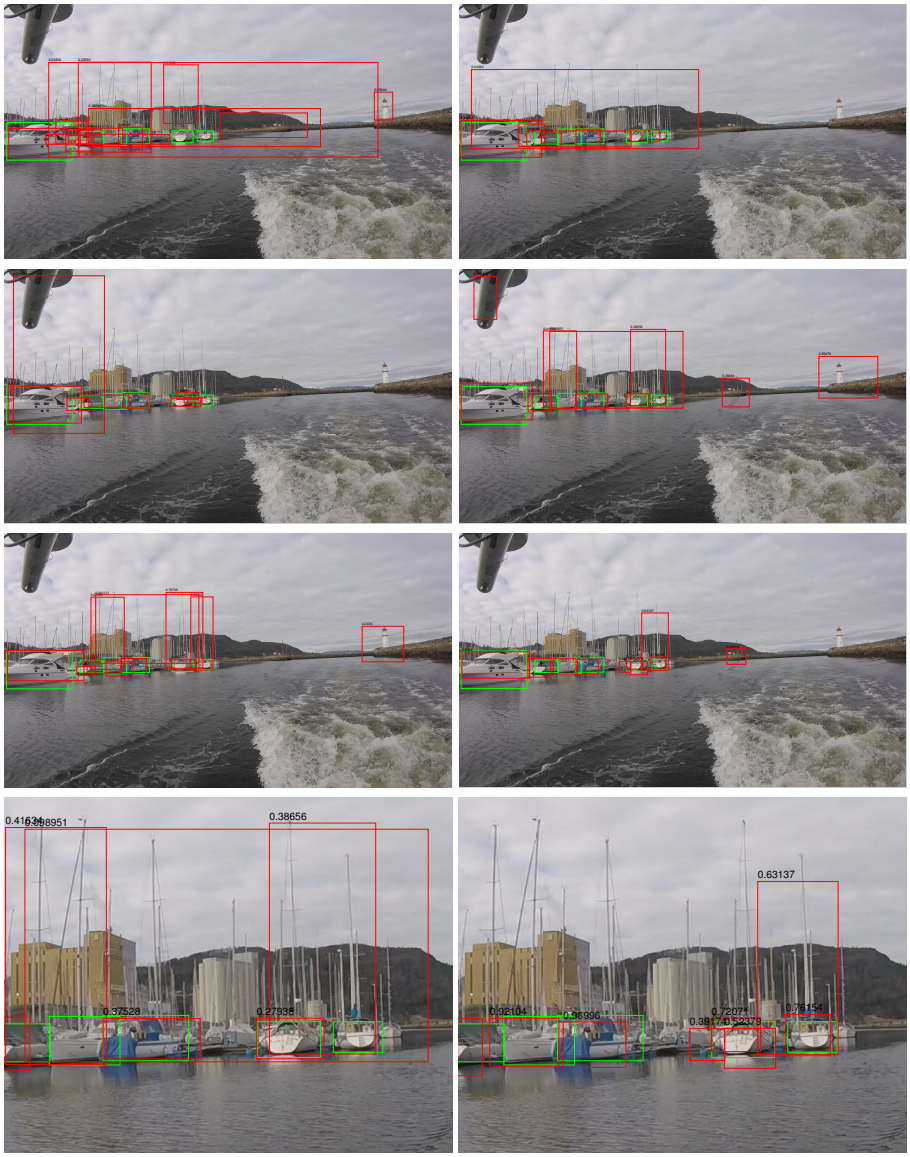
The exhibitions in Figure 8.12 and 8.13 were provided to give a final insight into how these models behave. Again, the E_4 model stands out with excellent performance. The other general models show a varying degree of precision. Some of these could have been considered positive predictions if the masts of the sailboats had been considered when drawing the ground-truth bounding-boxes in creating the data set. This was a conscious decision to avoid, as they induce a large amount of background-noise into the ground-truth boxes, and displace the bounding-box centres off the object. In Figure 8.13**F** the E_4 model has in fact outperformed the author, detecting ships that were not annotated, but were in fact ground-truth objects. A small brown vessel that is flush with the background has been detected at the left side of the image, as well as a larger docked vessel at the right of the image. This is also the case for Figure 8.13**H**, which is a zoom of Figure 8.13**G**. A vessel is detected at the right side of the image, in front of two annotated vessels.



A	**B**
C	**D**
E	**F**

A E_1. **B** E_2. **C** E_{3A}. **D** E_{3B}. **E** E_{3C}. **F** E_4.

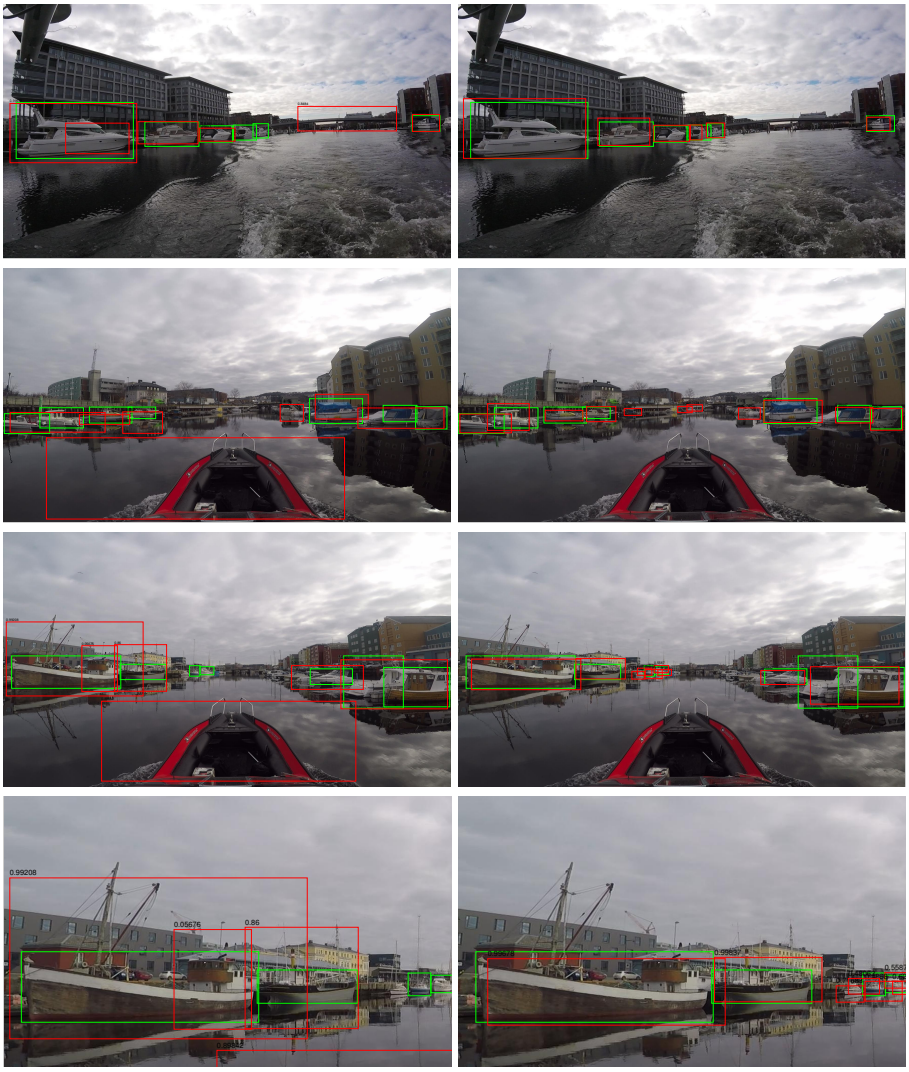
Figure 8.11: Distance error for True Positives on the Custom data set, all models.



A	**B**
C	**D**
E	**F**
G	**H**

A E_1. **B** E_2. **C** E_{3A}. **D** E_{3B}. **E** E_{3C}. **F** E_4.
G Zoom of E_{3B} output (**D**). **H** Zoom of E_4 output (**F**).

Figure 8.12: Output samples from the Custom test-set, all models.



A	**E**
B	**F**
C	**G**
D	**H**

A-D E_{3B}. **E - H** E_4.

Figure 8.13: Additional output samples from the Custom test-set for the default iteration VGG-16 model trained on VOC0712+Imagenet and the VGG-16 model trained on VOC0712+Imagenet+Custom.

8.6 Video Analysis

The results have shown promising performance for the single class object detectors. The two best performing detectors on the VOC2007 test-set is the E_{3B} and E_4 models, while the E_4 model showed a large performance increase in the custom test-set. This section analyses a video comparing the two detector's performance. This is complimentary to the rest of the results, and sectioned off for readers not able to view the video. The video can be found following this link:

https://youtu.be/gw7iG3ORMjg

Clip	Time span	Description
1	0:00 - 0:13	Approaching three docked ships in harbor
2	0:13 - 0:26	Approaching harbor. Docked ships in the right side of the image, occluded out of frame as video as ship approaches. Small grey dinghy starting in center of first frame and traversing to the left. Large ship in center of frame throughout the video.
3	0:26 - 0:40	In harbor. Docked sailboats in the right side of the image.
4	0:40 - 0:53	In harbor. Docked boats on both sides of the view.
5	0:53 - 1:06	Traversing under bridge. Docked boats entering and exiting view from both sides.

Table 8.6: Video Analysis: Video description

The video is partitioned into 5 different clips, a brief description of the content of these clips is given in Table 8.6. The video shows the E_{3B} model in the left window and the E_4 model in the right window. All the clips in this video is part of what constitutes the custom data set.

Discussion

Clip 1: At the start of the clip, the E_4 model has already detected two out of three of the docked ships. This points towards the models improved performance due to the training-images lying in close relation to the images in the video. The E_{3B} model gives a single correct detection at the end of the video enveloping the two closest ships. At the right side of the video, a building is shown. The E_{3B} model wrongly classifies this building as a ship at multiple occasions, while this is never the case for the E_{34} model. It is this that is the important difference in the models, especially in pursuit of a robust detector that can be used on a predefined path.

Clip 2: Both models correctly detect the ships exiting the view on the right. The E_{3B} model manages to excel here, detecting these ships in more frames than the E_4 model. For the large ship in the centre of the frame the E_4 model excels as it is part of a training-image. The small dinghy traversing from the center and to the left of the frame, both models struggle, as it is low in contrast and color. The E_4 model picks it up in some frames around 0:25 when it approaches the training-image where it was labelled as ground truth. What came

as a surprise here is that the E_4 model picks up more false positives from the surrounding than E_{3B}. Both models wrongly detect the buildings on the left of the frame. These buildings however, seem to have close similarity to a ship behind some docks. In addition to this, the E_4 model detects some red buildings throughout the clip at the middle-right of the frame. The reason for this is likely the abundance of white boats in the custom training-set.

Clip 3: A lot of correct detections are done for the E_4 model throughout the clip due to ground-truth objects appearing. The E_{3B} model shows some promising performance here, for the most part correctly detecting ships in the right area of the image. It does however wrongly detect some of the stone jetty on the left side of the frame at multiple instances.

Clip 4: Again, a lot of correct detections are done for the E_4 model throughout the clip due to ground-truth objects appearing. The E_{3B} model, while not detecting all the docked boats, generally have positive detections in the right areas of the image. No false positives being wrongly detected over a large amount of frames stand out in this clip, which is quiet remarkable, for both models.

Clip 5: The E_4 model performs well throughout the clip, correctly detecting most of the ships. The E_{3B} model incurs some false positive detections as the vessel is traversing under the bridge. This again illustrates the point of the E_4 model. To eliminate false positives done on the static environment. E_{3B} picks up a large amount of the ships at a close range, but struggles to detect vessels at a larger distance.

8.7 Summary

This thesis has investigated different means of achieving robust detection for the ASV application. This section gives a summary of the experiments conducted to achieve the results presented in this paper.

Experiment	Classes	Data sets	Data set size (Ims)	Relevant data (Ims)	Hardware
F_{ZF}	20	VOC0712-trainval	11540	689	-
F_{VGG}	20	VOC0712-trainval	11540	689	-
E_1	20	VOC0712-trainval	11540	689	Comp_1
E_2	20	VOC0712-trainval+Imagenet	13199	2348	Comp_1
E_{3A}	1	VOC0712-trainval+Imagenet	2348	2348	Comp_1
E_{3B}	1	VOC0712-trainval+Imagenet	2348	2348	Comp_2
E_{3C}	1	VOC0712-trainval+Imagenet	2348	2348	Comp_2
E_4	1	VOC0712-trainval+Imagenet+Custom	2448	2448	Comp_2

Table 8.7: Implementation and data sets

Experiment	Model	RPN Iterations		Fast R-CNN Iterations		VOC2007 test boat	Custom test
		$\eta = 0.001$	$\eta = 0.0001$	$\eta = 0.001$	$\eta = 0.0001$	AP (%)	AP (%)
F_{ZF}	ZF-Net	60k	20k	30k	30k	46	-
F_{VGG}	VGG-16	60k	20k	30k	30k	65	-
E_1	ZF-Net	60k	20k	30k	30k	49	7
E_2	ZF-Net	60k	20k	30k	30k	52	6
E_{3A}	ZF-Net	60k	20k	30k	30k	63*	11
E_{3B}	VGG-16	60k	20k	30k	30k	67*	17
E_{3C}	VGG-16	60k	20k	30k	10k	65*	17
E_4	VGG-16	60k	20k	30k	30k	67*	55

* Subset of VOC2007 test-set with 172 images containing the boat class.

Table 8.8: Results and training aspects

Table 8.7 displays the total data sets and the hardware used for training the different models. F_{ZF} and F_{VGG} denote the models published in the Faster R-CNN paper [25]. All VGG-16 models are trained on the comp_2 setup with a Nvidia Titan Xp GPU which is necessary to store all model-parameters in memory. Table 8.8 display the results from all experiments accompanied by the training aspects to each experiment. The AP for the custom test-set was not computed for the original models. This is due to the fact that it needs to be computed during the training-stage, and the Faster R-CNN repository does not contain the model-stages for the different models. While it is possible to accomplish testing outside of the training stages, heavy modification to the framework is necessary. Verifying the integrity of such a modification was deemed unnecessary, as E_1 performed virtually the same as F_{ZF}, which gave the possibility to test the model on the custom test-set. Thus, the expected performance of F_{ZF} is close or equal to that of E_1.

Experiment	Model	RPN training time (hours)		Fast R-CNN training time (hours)		Total training time	Hardware
		Stage 1	**Stage 2**	**Stage 1**	**Stage 2**		
E_1 - E_{3A}	ZF-Net	2	2	3	2	9	Comp_1
E_{3B} - E_4	VGG-16	4	2	7	4	17	Comp_2

Table 8.9: Training-stage durations

Part V

Closing Remarks

Chapter 9

Conclusion

This chapter concludes this thesis, summarizing the most important findings and presenting some future challenges.

- Section 9.1 gives a brief overview of the work presented in this thesis.
- Section 9.2 underlines the most important findings of the thesis.
- Section 9.3 presents some future work that might be worth investigating.

9.1 Overview

This thesis has investigated the application of a deep learning approach to ship detection. This work was carried out to provide robust detection with a optical sensor (camera) with the purpose of aiding maritime collision avoidance. The detector uses the state of the art object detection and object classification framework Faster R-CNN [25] as a starting-point. The framework has been modified by training new models and sampling new and relevant data for the application.

Regions of interest were defined in images using an RPN. The RPN provided 300 region proposals for a given image input. This was obtained using a CNN of arbitrary depth. The image and the regions of interest were then classified using Fast R-CNN. The same arbitrary CNN model as the RPN network was employed for model-parameter sharing. The features were subject RoI-pooling and ultimately a C-class softmax, determining the feature affiliation to the C classes defined by the user.

Data from several sources was used to train these object detection and classification networks. Large, general purpose datasets with images of ships and boats from imagenet and the VOC constituted the main data. In addition, a smaller, custom dataset was made. This provided training and testing images in close relation to each other.

The two CNN models investigated in this thesis were the 5 convolutinal layer deep ZF-Net model and the 16 layer deep VGG-16 model [34],[30]. These models were put through rigorous testing, being trained on the different data sets. The motivation has been to provide a heuristic for future development using this method of object detection for the ASV application.

9.2 Findings

The first experiment (E_1) was conducted as an attempt to recreated the results obtained in Faster R-CNN [25]. This was done for the ZF-Net model trained on the VOC0712 data set and achieved the same mAP as presented in Faster R-CNN. This provided a benchmark for future experiments, especially when it came to evaluation of the ASV application, as the training-stages were not provided in the Faster R-CNN repository [29]. Having obtained this benchmark, it was deemed unnecessary to do the same for the VGG-16 model as it would produce the same results.

Experiment (E_2) investigated how the results would change if the data set was padded with additional images of boats. As boats are rather poorly represented in the VOC data sets, it was hypothesized that an increased representation would provide improved overall performance. At its output, a 3% increased AP was achieved on the VOC2007 test-set, giving evidence for this hypothesis.

A large amount of redundant classes were present in the default 20-class network. It was only natural to investigate how the network would perform removing these. Experiment 3 (E_3) consisted of a set of sub-experiments, training the network on a single class. This was also the first time the network was trained on the VGG-16 CNN model. All sub-experiments were trained on the boat-images from the VOC data sets, as well as the imagenet data set. The best performing network used the VGG-16 model, trained on the full amount of iterations as it is presented in Faster R-CNN [25]. Unfortunately, heavy modification was necessary to test these on the full VOC2007 test-set, and they were therefore tested on a subset of these containing only images of ships. It was however later verified that the performance was superior in all E_3 models compared to the E_2 model of the same data set using the custom test-set.

In experiment 4 (E_4) a new data set was introduced to the training-regime. This data set was constructed for the ASV task, with images captured from a video-stream containing different scenarios that an ASV might encounter. The Custom data set included complex scenes from inside a harbor, having multiple stationary targets as well as at open sea with moving targets. The experiment introduced this data set to the best performing model of E_3 and evaluated it on the same test-set as E_3. The VOC2007 test-set AP did not increase particularly in this experiment. The E_4 model however outperformed all other models on the custom test-set, which was the evaluation for the ASV application. This indicates that a more local robust detector can be built with training-data taken from a predefined-path. This however, requires further investigation.

9.3 Future Work

Applying CNNs for object detection and classification is a field of heavy research, and every year improved methods appear. What this thesis has investigated is a state of the art approach at the time of writing. In the field of object classification, a CNN model with improved performance over the VGG-16 model was published in 2016 called ResNet-101 [18]. Resnet-101 is a 101 layer deep CNN, and surpassed VGG-16 with 3.2% mAP, trained on VOC0712 and tested on VOC2007. An attempted implementation was done of ResNet-101 for this thesis, but due to incompatibility with the Caffe version used it was discarded. All deploy files can be found in the Resnet repository which is available at the Caffe model zoo [3].

Concerning the findings in this thesis there are aspects which should be explored further. Building a larger and more diverse training and test data set is one thing that is necessary in order to evaluate the true robustness of such a detector. Collecting image data on the same path at different times of year with different targets would be ideal. In the case of evaluating robustness, not enough data can be obtained. Another possibility that should be explored is adding some background-classes to the training-data. Adding classes such as house, harbor and mountain to name a few, could help eliminate some of the false positive detections for the 1-class models. Imagenet provides synsets for all the above-mentioned classes.

In the tracking pipeline it is necessary to estimate the Cartesian coordinates of the detected vessels. A good first step is the extended Kalman filter estimation of Cartesian coordinates given image bounding-boxes and calibrated camera parameters [22]. The paper explores both simulated target estimation as well as real sea trials. A preliminary to this approach is a calibrated camera, as well as the pinhole camera model for relating world coordinates to the image plane [14].

Bibliography

[1] Caffe. Online Documentation. URL http://web.archive.org/web/20170101163210/http://caffe.berkeleyvision.org/.

[2] *Neural Networks and Deep Learning*. URL https://web.archive.org/web/20160516185148/http://host.robots.ox.ac.uk/pascal/VOC/.

[3] Model zoo. Online Repository. URL https://web.archive.org/web/20160713185549/https://github.com/BVLC/caffe/wiki/Model-Zoo.

[4] Convolutional neural networks (cnns / convnets). Online Lecture. URL https://web.archive.org/web/20161117211846/http://cs231n.github.io/convolutional-networks/.

[5] Pascal visual object classes. Online Repository, . URL https://web.archive.org/web/20160516185148/http://host.robots.ox.ac.uk/pascal/VOC/.

[6] The pascal visual object classes challenge 2012 (voc2012) development kit. Online Documentation, . URL http://host.robots.ox.ac.uk/pascal/VOC/voc2012/devkit_doc.pdf.

[7] Tonje Nanette Arnesen and Richard B. Olsen. Literature review on vessel detection. Technical report, Forsvarets Forskningsinstitutt, 2004.

[8] B.E. Bayer. Color imaging array, July 20 1976. URL http://www.google.com/patents/US3971065. US Patent 3,971,065.

[9] Christopher M Bishop. Pattern recognition. *Machine Learning*, 2006.

[10] George A. Miller Christiane Fellbaum. *WordNet: An Electronic Lexical Database*. 1998.

[11] Jia Deng, Wei Dong, R. Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2009.

[12] Li Dong, Li Yali, He Fei, and Wang Shengjin. Object detection in image with complex background. In *Proceedings of 3rd International Conference on Multimedia Technology*. Atlantis Press, 2013.

[13] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[14] David A Forsyth and Jean Ponce. A modern approach. *Computer vision: a modern approach*, pages 129–147, 2003.

[15] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1440–1448, 2015.

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

[17] Richard HR Hahnloser, H Sebastian Seung, and Jean-Jacques Slotine. Permitted and forbidden sets in symmetric threshold-linear networks. *Neural computation*, 15(3): 621–638, 2003.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[19] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[20] Alexander Kadyrov, Hui Yu, and Honghai Liu. Ship detection and segmentation using image correlation. In *2013 IEEE International Conference on Systems, Man, and Cybernetics*, pages 3119–3126. IEEE, 2013.

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[22] Dann Laneuville, Adrien Negre, and Pauline Dufour. 4d cartesian state estimation of sea surface targets with a single camera. In *2016 IEEE Aerospace Conference*. IEEE, 2016.

[23] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[24] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[25] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 91–99. Curran Associates, Inc., 2015.

[26] Raúl Rojas. *Neural networks: a systematic introduction*. Springer Science & Business Media, 2013.

[27] M Uma Selvi and S Suresh Kumar. A novel approach for ship recognition using shape and texture. *International Journal of Advanced Information Technology*, 1(5): 23, 2011.

[28] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[29] Ross Girshick Jian Sun Shaoqing Ren, Kaiming He. Faster r-cnn: Towards real-time object detection with region proposal networks. `https://github.com/ShaoqingRen/faster_rcnn`, 2016.

[30] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[31] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[32] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alex A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *ICLR 2016 Workshop*, 2016.

[33] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 104: 154–171, 2013.

[34] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013.

[35] Will Y Zou, Xiaoyu Wang, Miao Sun, and Yuanqing Lin. Generic object detection with dense neural patterns and regionlets. *arXiv preprint arXiv:1404.4316*, 2014.

Appendix

Modifying Faster R-CNN for your needs

This appendix is aimed towards any new user who want to train Faster R-CNN on their own dataset. These are the modifications done in this thesis for rearranging, and modifying the code to match the custom dataset.

If you wish to start training with Faster R-CNN outside the VOC datasets there are a couple of steps that can help you along the way. The answers to the questions that might arise are typically scattered around the repository of Faster R-CNN [29], or more general problems arise in caffe discussions on various online platforms. The rest of this appendix will assume that the user has followed the README.md file of the Faster R-CNN repository and meet the required specifications both for hardware and software. For instructions on rebuilding 'caffe for Faster R-CNN' refer to Section 6.3.2.

Preparing with VOC datasets

The easiest way to get started with Faster R-CNN is to download one or both of the VOC2007 and VOC2012 datasets [5]. The contents of these datasets need to follow the VOC hierarchy of the VOC documentation ([6] Section 7.1) with the root for the dataset being

```
.../faster_rcnn-master/datasets/VOCDevkit20XX
```

The image-sets and VOCcode should then be contained in

```
.../faster_rcnn-master/datasets/VOCDevkit20XX/VOC20XX/ImageSets
    .../faster_rcnn-master/datasets/VOCDevkit20XX/VOCcode
```

Using the VOC hiearchy as a template, the easiest way to create a new dataset is to create all folders contained in the VOC20XX dataset into a new location, given a new name. For future reference, the name *own* will be used to denote the new custom dataset. This dataset should then contain all folders as per [6] Section 7.1 such that the image-sets and VOCcode can be found in

```
.../faster_rcnn-master/datasets/OWN/own/ImageSets
   .../faster_rcnn-master/datasets/OWN/VOCcode
```

The use of capitalization is certainly up to the user.

Class modifications

VOC changes

Changes to classes and amounts of classes outside of Faster R-CNN can now be done for the new dataset.

`OWN/VOCcode/`

`VOCinit.m`

```
1
2 VOCopts.dataset='OWN';
3
4 % OWN classes
5
6 VOCopts.classes={...
7     'aeroplane'
8     'bicycle'
9     'bird'
10    'boat'
11    'bottle'
12    'bus'
13    'car'
14    'cat'};
```

Changes to `VOCopts.classes` is done per demand. It is important to keep tally on capitalization, making sure there is a case sensitive match between `VOCopts.classes` and the annotation name-tags.

Caffe changes

With the classes defined as above, the Faster R-CNN layers need to be changed to accommodate for the change in output dimension. All changes to the model layers is done in the deploy files in

`.../faster_rcnn-master/models/XXXXX_prototxts/`

The above folders hold the deploy folders for both the RPN and Fast R-CNN model stages. The underscore delimiter in the folder notation such as `ZF_fc6` implies the first layer subject to training using the files in this folder. In turn, the deploy folders hold the caffe `.prototxt` deploy-files denoted `train_val.prototxt` and `test.prototxt` aswell as the solver, which is not configured here.
Independent of which CNN model is used, the following changes need to be made to all deploy-files in all deploy-folders belonging to the model

train_val.prototxt

```
input:  "bbox_targets"
input_dim:  1
input_dim:  4*(K+1) for K classes
input_dim:  1
input_dim:  1
bottom:  "fcX"
name:  "cls_score"
       ⋮
inner_product_param {
       num_output:  K for K classes
bottom:  "fcX"
name:  "bbox_pred"
       ⋮
inner_product_param {
       num_output:  4*(K+1) for K classes
```

test.prototxt

```
bottom:  "fcX"
name:  "cls_score"
       ⋮
inner_product_param {
       num_output:  K for K classes
bottom:  "fcX"
name:  "bbox_pred"
       ⋮
inner_product_param {
       num_output:  4*(K+1) for K classes
```

Faster R-CNN changes

There are a couple of prerequisites left in the Faster R-CNN Matlab framework that needs to be addressed in order to start training on a custom dataset. Create a function in the same fashion as i.e. `voc2007_devkit.m` pointing to the OWN directory.

`faster_rcnn-master/experiments/+Dataset/private/`

`OWN_devkit.m`

```
function path = OWN_devkit()
    path = './datasets/OWN';
end
```

Further changes are needed to correctly create a cache database (imdb) from the images and annotations in the OWN dataset. The function below shows the modifications dont to `imdb_from_voc.m`

`faster_rcnn-master/imdb/`

`imdb_from_OWN.m`

```
function imdb = imdb_from_OWN(root_dir, image_set, flip)
%This code snippet shows the necessary modifications to
%faster_rcnn-master/imdb/imdb_from_voc.m
cache_file = ['./imdb/cache/imdb' '_' image_set];

imdb.name = image_set;
```

Having created the necessary functions, the new dataset can be defined using a modification of `VOCXXXX_trainval`

`faster_rcnn-master/experiments/+Dataset/`

`OWN_trainval.m`

```
function dataset = OWN_trainval(dataset, usage, use_flip)
% OWN trainval set
% set opts.imdb_train opts.roidb_train
% or set opts.imdb_test opts.roidb_train

OWN                         = OWN_devkit();

switch usage
    case {'train'}
        dataset.imdb_train    = {imdb_from_OWN(OWN, '
            trainval', use_flip)};
        dataset.roidb_train   = cellfun(@(x) x.roidb_func(x
            ), dataset.imdb_train, 'UniformOutput', false);
    case {'test'}
        error('only supports one source test currently');
    otherwise
        error('usage = ''train'' or ''test''');
end
end
```

The dataset is now ready to be used in any training with an arbitrary model. It is loaded the same way as the VOC datasets following the modifications to any training routine

`faster_rcnn-master/experiments/`

`script_faster_rcnn_OWN_XXX.m`

```
%Modifications to script_faster_rcnn_VOCXXXX_XXX.m

dataset                     = [];
dataset                     = Dataset.OWN_trainval(dataset,
    'train', conf.use_flipped);
```