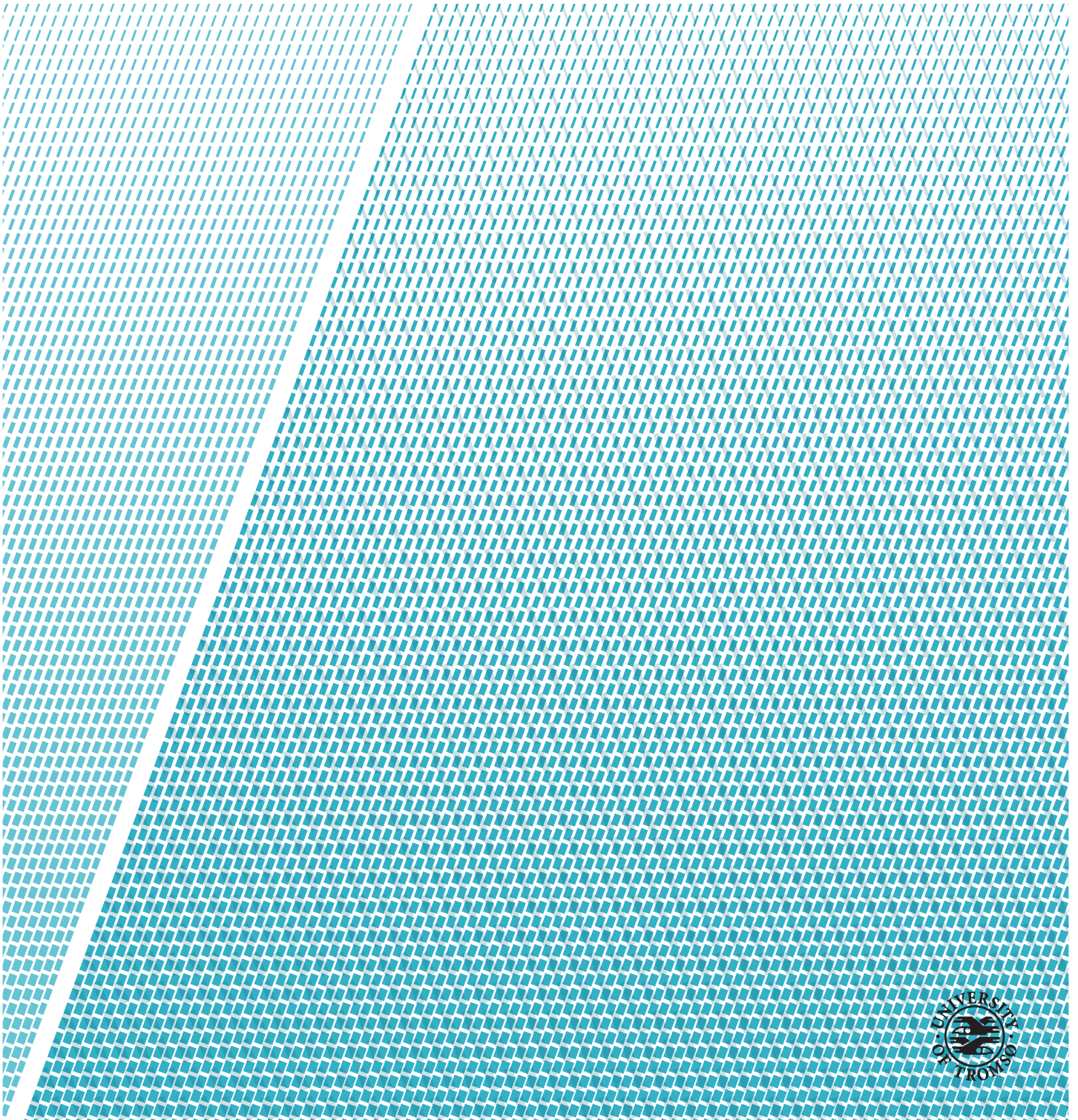


Deep Learning of Oriented Bounding Box Regression Networks for Ship Detection in Optical Satellite Images

Åsmund Mikael Sandland

FYS-3941 Master's thesis in applied physics and mathematics - 30 stp - June 2020



Abstract

Maritime surveillance is important for management of maritime traffic and to prevent activities like illegal fishing, hazardous cargo transportation, piracy, and smuggling of goods and humans. Remote sensing is frequently used for positioning vessels that are not transmitting via the Automatic Identification System (AIS). Modern optical remote sensing instruments provide high-resolution imagery, allowing for advanced analyses of the Earth's surface.

Human operators are trained to recognize different structures and objects in satellite images, resulting in precise scene analyzes. This endeavor is, however, time consuming and expensive, and the earth observation community is continuously researching how to effectively and precisely automate this process. For ship detection in remote sensing images, state-of-the-art architectures are based on deep neural networks. This thesis contributes data, experiments and architectures that are based on deep neural networks. Recognizing vessel heading may be useful for assessing its intentions, and is an interesting topic in the application field which will be studied in this thesis. This is obtained by deep learning of regression networks that assign rotated bounding boxes to detected vessels.

A data set with high-resolution SuperView optical satellite imagery and rotatable bounding box annotations is contributed by this thesis. Experiments on five reference object detectors are performed, giving results on the reliability of running ship detection services on SuperView images. Scenes of high object density are studied. Explicitly, experimental results on the newly proposed Object Detection with Grouped Instances (ODGI) (Royer and Lampert, 2020) show slightly increased performance when utilizing grouped object instances, compared to equivalent models that use individual object instances.

The novel Oriented YOLOv2 and Oriented Tiny YOLO neural network architectures, extending from YOLOv2 (Redmon and Farhadi, 2017) and Tiny YOLO, recognize object rotation and provide a more accurate shape description than the predecessors. These are used in the novel Oriented Object Detection with Grouped Instances (OODGI) pipeline, extending from the newly proposed ODGI (Royer and Lampert, 2020), to utilize object groupings while providing

positioning, shape and rotation predictions. An additional error analysis of 11 reference and novel neural network architectures is supplemented to study model sensitivities. Experiments on performance consistency of deep neural network architectures when the amount of training data is limited reveal that resulting precision varies over different training sessions. This variation is discussed to be induced by stochasticity in weight initialization and batch selection.

The experimental results indicate that Faster R-CNN has the highest precision. However, ODGI is three times faster and has competitive precision. The novel models proposed in this thesis successfully describe positioning, shape and orientation of ships, although OODGI needs some amendment.

Acknowledgements

I would like to express my gratitude to my helpful and talented supervisors Stian N. Anfinsen and Torgeir Brenn for guiding and motivating me through this thesis. Thank you Stian for sharing your best advice and experiences, for helping me stay focused and including relevant topics, and for countless hours of reviewing and correcting this thesis. Thank you Torgeir for providing this interesting and fascinating subject, for your theoretical guidance, and for resolving the data and computer setup related issues.

The rest of the UiT Machine Learning Group have been helpful with providing computational machinery, and contributing deep learning expertise when necessary. Thank you KSAT for showing understanding and help out with data and computational resources in these extraordinary times, and for guidance related to the data.

Thank you to all my beloved friends. My childhood friends for always providing a good mood, and for countless pleasurable hours of talking. My dear fellow students, especially you Sander and Joakim, for all the shared beers and enjoyable conversations during busy times, thank you.

Lastly, I would like to express my sincerest gratitude to my mom, my dad and my brother Kristian, and the rest of my family. You are always there for me, offering supportance, and are the reason I have been able to complete this Master's degree.



"The harder you work for something, the greater
you'll feel when you achieve it"

Contents

Abstract	i
Acknowledgements	iii
List of Figures	viii
List of Tables	ix
List of Algorithms	ix
List of Abbreviations	x
I Introduction	1
1 Ship Detection using Remote Sensing	3
2 Thesis Formalities	7
2.1 Hypotheses	7
2.2 Thesis Outline	8
2.3 Thesis Structure	8
2.4 Mathematical Nomenclature	9
II Theory and Related Work	11
3 Remote Sensing	13
3.1 Remote Sensing Principles	13
3.2 Resolution	14
3.3 Optical Satellite Images	14
3.3.1 Pansharpening	15
3.3.2 SuperView	15
3.4 Image Tiles	16

4	Machine Learning	17
4.1	Supervised and Unsupervised Learning	18
4.2	Neural Network	19
4.2.1	Activation Function	20
4.3	Convolutional Neural Networks	21
4.3.1	Convolution	21
4.3.2	Pooling	21
4.3.3	Transposed Convolution	22
4.3.4	Convolutional Neural Networks	22
4.4	Optimization Problem for Supervised Learning	23
4.4.1	Parameter Optimization	24
4.4.2	Objective Function	24
4.5	Optimization Methods	25
4.5.1	Gradient Descent	26
4.5.2	Momentum and Adam	28
4.6	Regularization	29
4.6.1	Overfitting	30
4.6.2	Weight Regularization	30
4.6.3	Batch Normalization	31
4.6.4	Dropout	31
4.6.5	Data Augmentation	32
4.7	Performance Measures	32
4.7.1	Common Measures	33
5	Object Detection using Deep Learning	35
5.1	Segmentation-based Models	37
5.2	Bounding Box-based Models	38
5.2.1	Non-maximum Suppression	38
5.2.2	Proposal-based Methods	39
5.2.3	Grid-based Methods	42
5.2.4	Rotation Invariant Networks	44
5.2.5	Rotatable Bounding Boxes	45
5.3	Deep Learning Computation	46
5.3.1	Pretrained Deep Learning Models	46
6	Object Detection Neural Networks and Ship Detection	47
6.1	Traditional Ship Detection	48
6.2	Architectures Based on Segmentation	49
6.3	Architectures Based on Bounding Boxes	49
6.3.1	Proposal-based Methods	50
6.3.2	Grid-based Methods	55
6.3.3	ODGI: Royer and Lampert (2020)	62
6.3.4	Summary	68

III	Methodology	69
7	SuperView Data Set	71
7.1	SuperView Appearances	73
7.2	Data Imbalance	74
7.3	Augmenting Optical Satellite Images	75
8	Novelties	77
8.1	Technical Problems to Address	78
8.2	Oriented YOLOv2	79
8.3	Oriented Tiny YOLO	83
8.4	Oriented Object Detection with Grouped Instances	83
IV	Experiments and Experimental Setup	88
9	Experiments on Reference Models	91
9.1	Experimental Setup	91
9.1.1	Faster R-CNN:	92
9.1.2	DRBox	93
9.1.3	YOLOv2	94
9.1.4	Tiny YOLO	95
9.1.5	ODGI	95
9.2	Experimental Results	97
10	Experiments on Novel Methods	103
10.1	Experimental Setup	103
10.1.1	Oriented YOLOv2	104
10.1.2	Oriented tiny-YOLO	104
10.1.3	OODGI	104
10.2	Experimental Results	105
11	Additional Experiments on Main Models	111
11.1	Error Analysis on Main Models	111
11.2	Stochasticity Analysis	113
V	Discussion and Conclusion	115
12	Discussion	117
12.1	Speed versus Accuracy Trade-Off	117
12.2	Discussion of Results	118
12.2.1	Reference Models	118
12.2.2	Novel Models	119

12.2.3 Additional Analysis	121
12.3 General	122
13 Concluding Remarks	125
13.1 Future Work	127
14 Appendix	129
14.1 RBox to BBox Mapping	129
14.2 Crop-Relative Annotation Mapping	130
Bibliography	131

List of Figures

1.1 Harbor scene with ground truth annotations	6
4.1 Conceptual sketch of how learning rate affects optimization .	27
4.2 Conceptual sketch of an overfitted optimization process . . .	30
4.3 Conceptual sketch of the dropout strategy	32
5.1 R-CNN architecture	41
5.2 YOLO architecture	43
5.3 SSD architecture	44
5.4 BBox and RBox comparison	45
6.1 Faster R-CNN architecture	52
6.2 DRBox architecture	56
6.3 ODGI architecture	63
6.4 Group object instance annotation	65
7.1 SuperView appearances	73
7.2 SuperView data augmentation	76
8.1 Group object instance annotation when using RBoxes	84
9.1 Annotation predicted using Faster R-CNN	98
9.2 Annotation predicted using DRBox	99
9.3 Predicted ships in harbor using reference models	100
9.4 Predictions in area of high ship density using reference models	101
10.1 Predictions in basic scene using novel models	107
10.2 Predictions on larger ships using novel models	108
10.3 Predictions in challenging scene using novel models	109
10.4 Predictions in harbor scene using novel models	110
11.1 Accuracy rates for error analysis	112
11.2 AP results from different training sessions	113

List of Tables

2.1	Mathematical nomenclature	10
3.1	SuperView bands and their properties	16
5.1	Comparison of benchmark object detection architectures . .	42
6.1	YOLOv2 layers	59
6.2	tiny-YOLO layers	62
9.1	Experimental results on reference models	98
10.1	Experimental results on novel models	106
11.1	Accuracy rates for error analysis	112

List of Algorithms

1	Adam optimization algorithm	29
---	---------------------------------------	----

Abbreviations

AI Artificial intelligence

AP Average Precision

BBox Bounding box

CNN Convolutional neural network

EO Earth observation

fps Frames per second

i.a. inter alia (latin for "among other things")

i.e. id est (latin for "that is")

IoU Intersection-over-Union

mAP Mean average precision

MS Multispectral

NMS Non-maximum suppression

NN Neural network

PAN Panchromatic

RBox Rotatable bounding box

ReLU Rectified Linear Unit

RoI Region of Interest

RPN Region proposal network


SAR Synthetic aperture radar

Part I

Introduction



Ship Detection using Remote Sensing

☞  As a core component in the scientific field of earth observation (EO), ship detection has been an attractive and necessary contribution. Object detection, and hence also ship detection, has made remarkable progress in recent years through the adoption of deep learning. Analyzing EO images using deep learning for detecting ships is the state-of-the-art, and is the essence of what is studied in this thesis.

Maritime surveillance is important for safe ship traffic and to control undesirable activities like illegal fishing, hazardous cargo transportation, piracy, and smuggling of goods and humans (Sandland, 2019). Most larger ships are required to transmit data via the automatic identification system (AIS)¹, and a large number of fishing boats and pleasure crafts use this system as well. AIS data are available for nearby ships to avoid collisions or in case of emergencies, and is a strong and widely used tool to regulate maritime traffic. However, ship owners with illegal intentions tend to manipulate the reported information or

1. The regulations are somewhat more intricate. The curious reader is encouraged to read the regulations defined by the Norwegian Coastal Administration:
<https://www.kystverket.no/Maritime-tjenester/Meldings--og-informasjonstjenester/AIS/AIS-regelverk-og-brukarkrav/>

simply turn off their AIS transmitter, resulting in complications when using this system as an information source (Balduzzi et al., 2014).

Offshore surveillance can be effectively automated, which it has been for many years, by the use of satellite remote sensing. Synthetic aperture radar (SAR) is traditionally used, and is still widely used, for the ship detection task as radar signals are mostly unaffected by weather conditions (Bamler and Hartl, 1998), and because the methods have been refined to, for instance, determine the size and velocity of a ship (Dragosevic and Vachon, 2008). SAR has a wide swath, and its microwaves are less affected by clouds and weather conditions, resulting in satellite images covering large footprints at all weather conditions. However, the spatial and temporal resolution of SAR is often considered insufficient to cope with many real world problems today (Liu et al., 2017b). Optical satellite imagery provides multiple high-resolution bands, which may be exploited to get details of smaller objects (Sai et al., 2019). Optical satellite imagery, and specifically SuperView images, are the remote sensing data used for ship detection experimentation in this thesis.

Kongsberg Satellite Services (KSAT) provides the raw data used for experimentation in this thesis. This includes SuperView optical satellite products, and a selection of annotated ships. The annotations have been adopted to a suitable object detection format and a complete training data set has been developed as a contribution by this thesis. KSAT provides vessel detection services to customers worldwide². Human operators are trained to recognize different structures and objects in satellite images, resulting in precise analyses of the scenes. Monitoring vast ocean areas using this endeavor is, however, time consuming and expensive, and the EO community is continuously researching how to effectively and precisely automate this process. Systems for this purpose have been up and running for decades, and they have been constantly evolving. Reliable detectors and data handling are essential for the daily work at KSAT. Ship detection in high-resolutional optical satellite imagery is a modern field at KSAT. This thesis is intended to study the opportunities that lie in optical remote sensing, and may contribute to optical remote sensing being adopted as part of KSAT's vessel detection services.

Machine learning, and particularly deep learning, has in the later years provided state-of-the-art EO systems. Convolutional neural networks (CNNs) are the quintessential deep learning models, the main cause of the tremendous progress, and can be adapted to fit various problems (Goodfellow et al., 2016). When a CNN is trained on appropriate training data, it has proved to perform better than traditional algorithms in a variety of computer vision and image

2. The curious readers are encouraged to immerse themselves in KSAT's vessel detection services at <https://www.ksat.no/services/earth-observation-services/>

analysis problems (Makantasis et al., 2015). Knowledge about the machine learning architecture and how it respond to different data is a necessity, and allows the opportunity of analyzing possible sources of errors. Enlightenment of underlying challenges in the system and data is desired. One possible challenge in optical EO data is appearance of small clouds. These may look very similar to ships and hence cause false alarms. Again, this risk can be mitigated by using a large amount of precise training data for optimization. The CNN can then learn to ignore these false alarms.

Rotated objects and heading prediction are of particular interest in this study. Knowledge on vessel heading can be useful for assessing its intentions. Even though object orientation is attractive in many applications, there is limited research work on object detection focusing on this problem (Liu et al., 2018). This is largely due to limited data sets labeled with rotation. Popular benchmark data sets, such as Pascal VOC (Everingham et al., 2010), ImageNet (Russakovsky et al., 2015) and MS COCO (Lin et al., 2014), have no information on object orientation. Airbus have published a data set concerning ship detection in optical satellite images (Kaggle, 2018), with a corresponding ship detection challenge, which has recently contributed to the interest in this field.

The scope of this thesis is to analyze properties of heading prediction and predicting objects in areas of high object density using bounding box-based predictors on high-resolutional optical remote sensing imagery. Concretely, the thesis will experiment on five reference deep learning models, some well established and some newly proposed methods, and three novel deep neural network architectures for object detection will be presented. The novel architectures are based on existing methods, with an extension concerning rotation recognition. Experiments on the novel models are performed to study their precision, speed, error properties and consistency. Scenes of high object density are of particular interest. The recently published Object Detection with Grouped Instances (ODGI) (Royer and Lampert, 2020), and the novel extension Oriented Object Detection with Grouped Instances (OODGI), will be used to study such scenes of high object density.

The motivation for studying scenes with high object density is visualized in Figure 1.1. This exemplifies a challenging scene encountering problems for most object detectors. By adopting grouped instances, improved precision in such scenarios are studied. By adopting rotatable annotations³, a visually more prominent description is desired. The scene in Figure 1.1 is really complex and improved performance in such areas is a goal. No object detectors are expected to process such scenes perfectly.

3. Objects in Figure 1.1 are annotated using rotatable bounding boxes.

As introduced earlier, the SuperView data set itself is considered a contribution by this thesis. As these SuperView images have not been analyzed for ship detection before, the experimentation using reference models is considered a contribution by this thesis. The novel neural network architectures and corresponding experimentation are also a contributing element.

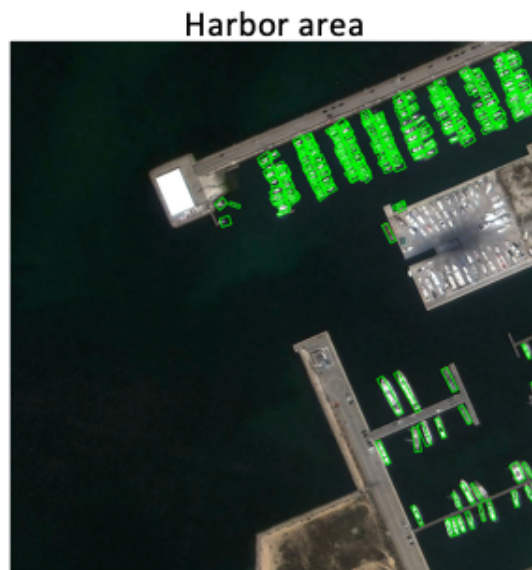


Figure 1.1: Visualization of an example SuperView segment where the object density is high. Ground truth annotations are supplemented. This summarizes the motivation for studying grouped object instances and rotated annotations.

/2

Thesis Formalities

Before approaching the essential parts of the thesis, some formalities must be clarified. This chapter will present three hypotheses, state the outline of the thesis, and specify the thesis structure.

2.1 Hypotheses

This thesis concerns ship detection in optical remote sensing images. Deep neural networks are used for vessel recognition. Prediction of ship rotation and utilization of object groupings for increased performance are topics of special interest. Three hypotheses to be answered in the thesis have been formulated:

Hypothesis 1. *Annotating objects in remote sensing images using rotatable bounding boxes gives technical improvements leading to increased precision over traditional bounding boxes for deep learning models, and gives a visually more prominent description.*

Hypothesis 2. *Object groupings are visually more salient and are easier detected than individual objects, leading to increased precision when applying deep learning architectures based on object groupings to the SuperView data set.*

Hypothesis 3. *Expanding deep learning architectures based on grouped object instances initially predicting traditional bounding boxes to now predict rotatable bounding boxes will describe objects more orderly, while safeguarding the advantages of utilizing object groupings.*

2.2 Thesis Outline

This thesis contributes the following elements:

- A complete high-resolution SuperView optical remote sensing data set suited for deep learning applications is finalized.
- KSAT are provided with an analysis of how different object detectors perform on the optical satellite images from a new sensor.
- An analysis and verification of the newly proposed ODGI (Royer and Lampert, 2020) is presented.
- Three novel neural network architectures predicting positioning, shape and rotation of objects are presented and experimented on.
- The ability of recognizing object rotation as a direct regression approach is studied.
- An analysis of speed versus precision for several object detectors is completed.
- A study of consistency of deep neural network architectures that are trained and evaluated using data set of limited extent is presented.

2.3 Thesis Structure

This thesis is organized in five parts, which are again divided into chapters. The chapters are divided into sections, and subsections where this is considered expedient.

Part I consists of chapter 1 concerning ship detection using remote sensing, and chapter 2 expressing thesis formalities. Chapter 1 introduces the concept of ship detection, its benefits to society, KSATs vessel detection services and the motivation behind the analyses of this thesis. Chapter 2 states the scientific hypotheses, contributing elements, and the organization of the thesis.

Part II includes detailed theory of relevant concepts and a review of related work. It consists of chapter 3 concerning remote sensing theory, chapter 4 describing machine learning and deep learning concepts, chapter 5 describing relevant theory of modern object detection, and finally chapter 6 describing work related to ship detection. Chapter 3 describes main concepts of remote sensing, optical satellite images, and the SuperView satellites are introduced. Chapter 4 carefully reviews neural networks, optimization of deep learning architectures, regularizing strategies and performance measures. State-of-the-art object detectors and methods get a conceptual introduction in chapter 5, before the relevant detectors are resumed and described in detail in chapter 6. Reference models are also explained in detail in this chapter.

Part III presents the SuperView data set in chapter 7 and novel architectures in chapter 8. Chapter 7 states the main properties of the data set, how it is collected, and potential challenges are discussed. When novel architectures are presented in chapter 8, all complications and necessary modifications are first described, before an explanation of the solutions are given where it is relevant for the three novel neural network architectures: Oriented YOLOv2, Oriented Tiny YOLO and OODGI.

Part IV presents experimental setup and results for all reference and novel architectures, separately. Chapter 9 describes the experimental setup of the models Faster R-CNN, DRBox, YOLOv2, Tiny YOLO and ODGI, with different complexities, which are the reference models used in this thesis. Results are supplemented. Corresponding description of experimental setup and results are given for the novel architectures in chapter 10. Chapter 11 provides an additional accuracy analysis on selected models, and stochasticity of different training sessions are studied.

Part V provides a discussion and conclusion in chapter 12 and 13, respectively. Chapter 12 discusses observed results from part IV, how the speed versus accuracy properties unfolds for the different detectors, and which factors to consider when selecting a ship detector. Chapter 13 offers some concluding remarks and ideas for future work.

2.4 Mathematical Nomenclature

The mathematical notation in the scientific fields of physics, statistics and mathematics, which are all contributing this thesis, varies widely. Expressions inspired by other references are translated to follow a common notation when used in this thesis. Table 2.1 summarizes the notation used in this thesis.

Table 2.1: This table defines the mathematical nomenclature in the thesis. Notations are separated into categories.

Numbers and Arrays	
a	A scalar
\mathbf{a}	A vector
\mathbf{A}	A matrix
\mathbf{o}	A vector having all elements equal to zero
Sets and Indexing	
\mathbb{R}	The set of real numbers
\mathbb{N}	The set of natural numbers. $\mathbb{N} = \mathbb{N}_1 = \{1, 2, \dots\} \subset \mathbb{R}$
\mathbf{a}_i	Element i of vector \mathbf{a} , with indexing starting at 1
$\mathbf{A}_{i,j}$	Element in matrix \mathbf{A} at row i and column j
$\{0, 1, \dots, n\}$	Set containing all integers between 0 and n
$\{\mathbf{x}^{(i)}\}_{i=1}^N$	Set containing elements of \mathbf{x} with index between 1 and N
$(a, b]$	Real interval excluding a but including b
Calculus and Linear Algebra	
$\nabla_{\mathbf{x}} y$	Gradient of y with respect to \mathbf{x}
$\sum_{i=1}^n a_i$	Sum of elements in \mathbf{a} having index between 1 and n
$\sum_i a_i$	Sum of all valid elements in \mathbf{a}
$\ \cdot\ $	Euclidean distance
$\mathbf{A} \odot \mathbf{B}$	Element-wise (Hadamard) product of \mathbf{A} and \mathbf{B}
\mathcal{O}	Order of a function. $\mathcal{O}(n^2)$ is quadratic order
Functions and Statistical Theory	
$\log(x)$	Natural logarithm of x
$\exp(x) = e^x$	Exponential of x
$\sigma(x)$	Logistic sigmoid. $\sigma(x) = \frac{1}{1+\exp(-x)}$
\circ	Function composition. $(g \circ f)(x) = g(f(x))$
$\%$	Modulus. $a \% b = a - \left\lfloor \frac{a}{b} \right\rfloor * b$
$*$	Convolution. The discrete convolution operator is defined in equation 4.4
$A \cap B$	Union of A and B . Sum of all elements in set A and set B
$A \cup B$	Intersection of A and B . Elements in set A also included in set B
$\mathbb{E}_{X \sim p}[\cdot]$	Expectation with respect to a stochastic variable X from a distribution p
Data Sets	
$\mathbf{x}^{(i)}$	i -th example (sample) from data set
$\mathbf{y}^{(i)}$	Target (label) associated with $\mathbf{x}^{(i)}$ in supervised learning
\mathbb{D}	Set containing the complete training data
$\mathbb{B} \subseteq \mathbb{D}$	Subset of the complete training data set. A batch

Part II

Theory and Related Work

/ 3

Remote Sensing

The formal definition of remote sensing is to apply recording devices that are not in physical, intimate contact with the items under surveillance - but at a finite distance from the observed target (Campbell and Wynne, 2011). Throughout this thesis, the term remote sensing will refer to remote sensing from satellites, and remote sensing images refers to the corresponding satellite images. Such images will be used in machine learning models for automatically detecting ships within the scene.

3.1 Remote Sensing Principles

The most widespread sensors in the field of remote sensing from satellites are synthetic aperture radar (SAR) and optical sensors. SAR operates with microwaves, and how the different polarizations are reflected can be analyzed to characterize the surface. Optical sensors operate in the visible and infrared parts of the electromagnetic spectrum, and are intuitive to interpret. SAR transmits energy and records the reflected energy (active sensor), and uses the echo time and Doppler frequency to position the reflections. Optical sensors records naturally emitted energy (passive sensor), typically from the sun. This restricts optical remote sensing instruments to only operate during day-time and in sufficient weather conditions. Optical sensors are also obstructed by clouds. This makes SAR a stronger tool for all-day, all-year monitoring. To increase the footprint recorded by optical sensors, the use of different *incidence angles* is expedient. The incidence angle is the instrument sensing direction when recording, and is corrected for in the produced satellite images.

For all satellite images, the sensed data is *discretized* to achieve image properties before it is transmitted to a ground station where it is processed and interpreted according to some objective.

3.2 Resolution

For remote sensing images, there are four relevant resolution terms describing the main properties of the instrument: *temporal resolution*, *spatial resolution*, *spectral resolution* and *radiometric resolution*. Temporal resolution is generally the time between reoccurring measurements. The temporal resolution for remote sensing refers to the revisit time of a satellite: how long time the satellite uses to return to the same geographical location (Campbell and Wynne, 2011). Spatial resolution for remote sensing refers to the spacing between pixels. If an image is acquired using 1 m spatial resolution, each pixel will describe a square with sides equal to one meter. Spatial resolution highly affects the appearance of an image and the size of objects present in the image. Based on this, machine learning models, which will be presented in chapter 4, trained on images acquired using a given spatial resolution cannot directly be generalized for inference on images acquired with another spatial resolution. Spectral resolution specifies the number of spectral bands a sensor is recording. Radiometric resolution describes the amount of discrete recorded intensities that can be stored and distinguished.

3.3 Optical Satellite Images

Optical satellite images are acquired by a multispectral imager (MSI). Campbell and Wynne (2011) presents an excellent overview of this technology: MSIs record multiple bands of different wavelengths, typically ranging from visible and near-infrared (VNIR) to short-wavelength infrared (SWIR). Each spectral band observes a small range of frequencies, and the measurement is a summation of all signals sensed within this range (Sandland, 2019).

Optical satellite images can typically be presented using only the red, green and blue frequency bands in an RGB or false colour image, but the near-infrared and infrared bands are also useful for monitoring different types of objects and surfaces. This thesis experiments on detecting ships in optical satellite images, where SAR images traditionally have been used. SAR measurements are sensitive to materials of high dielectric constant, making metallic ships easy to detect (Sandland, 2019). Optical images typically have a higher spatial resolution and increasing the potential of picking up finer details and smaller ships.

3.3.1 Pansharpening

A traditional technique is to record one panchromatic (PAN) band in high resolution, and numerous lower-resolution multispectral (MS) bands in different wavelengths¹. The PAN band typically records wavelengths ranging over the total wavelength range of the MS bands, while each MS band individually records over a smaller range of wavelengths. By utilizing the high-resolution PAN band, lower-resolution MS bands may be transformed to high-resolution variants as well. This technique, called pansharpening, is a widely used technique applied in a various scenarios. Google Earth exploits this principle to provide an enhanced result to the users while restricting the storage and processor usage (Vivone, 2014).

In practice, there are different techniques for achieving the pansharpened result. Generally, the pansharpening methods follow the following protocol: 1) From the PAN band, extract the high-resolution geometrical details that are not present in the MS band; 2) incorporate this geometrical details into the lower-dimensional MS bands by properly modeling the similarities in the PAN band and the MS bands (Vivone et al., 2014). In the later years, it has also been experimented on using neural networks (as will be introduced in chapter 4) for pansharpening (Masi et al., 2016).

3.3.2 SuperView

Data used in this thesis is collected from SuperView-1 satellites (often referred to as only SuperView). The SuperView satellites are China's first optical satellite constellation with a spatial resolution down to 0.5 m, operated by Beijing Space View Tech Co Ltd (Sai et al., 2019)². At present, it consist of four identical sun-synchronously orbiting optical remote sensing satellites: SuperView-1 o1 and SuperView-1 o2 launched in 2016, and SuperView-1 o3 and SuperView-1 o4 launched in 2018. Each satellite uses 97 minutes per orbit and in total they operate with a temporal resolution of only one day, making the constellation suited for a daily EO change monitoring. The project is still developing and the constellation is estimated to be complete in 2022, having a *daily acquisition capacity* of 12 million km² ³. Daily acquisition capacity is currently 3 million km².

1. Using one panchromatic band to increase the spatial resolution of the other bands is a cheap alternative to all bands recording in high resolution.
2. Word on the street is that SuperView is a Chinese imitation of the American WorldView constellation with a spatial resolution down to 0.31 m (Longbotham et al., 2015).
3. Daily acquisition capacity is the recorded footprint from a satellite pr. day (Campbell and Wynne, 2011).

Table 3.1 presents the main properties of the five bands recorded by sensors on each of the four SuperView satellites. Notice the wide wavelength range of the PAN band, and the difference in spatial resolutions. This can be utilized to perform pansharpening.

Table 3.1: SuperView-1 bands with corresponding wavelength-range and spatial resolution. Data provided by Sai et al. (2019).

Band number	Band name	Wavelengths [nm]	Spatial resolution [m]
1	Panchromatic	450-890	0.5
2	Blue	450-520	2
3	Green	520-590	2
4	Red	630-690	2
5	Near-IR	770-890	2

3.4 Image Tiles

Satellite images are of large scale containing a vast amount of information. A SuperView satellite image have lengths of $\sim 3.2 \times 10^4$ pixels, and occupies ~ 3 GB of storage capacity. It is therefore convenient to split it into subimages, called *tiles*. Together, the tiles fully represent the satellite image, while being more manageable. The tile size can be determined to fit the specific problem and model. A drawback of employing tiles, is that objects and important details may end up on the split. A common approach is to have some overlap, and to ignore details on the far edge when evaluating the image.

/4

Machine Learning

Machine learning as a scientific field has been around for decades, but has shown outstanding results for various applications in particular in the later years (Alpaydin, 2014). The accessibility of data at academic and private level has resulted in a wide interest in the field, with subsequent magnificent architectures for interpreting and learning upon this data. Efficient and convenient programming frameworks is another beneficial result of the wide interest in the field.

Larger machine learning models, specifically deep learning architectures¹, consist of millions of parameters needed to be properly adjusted for correct decision making. *Training* such a model, that is, adjusting these parameters to fit the specific problem, is a comprehensive and computer-intensive procedure. Graphics processing units (GPUs) have played a key role in the success of expanding machine learning architectures into more complex deep learning models (Shi et al., 2016). GPUs makes it possible to run real-time decision making, and have drastically reduced the training- and inference time of machine learning architectures, over the former central processing units (CPUs).

A machine learning model is trained using an optimization technique, as will be presented in section 4.4, to best solve the problem relative to the training data. A loss function, as will be presented in the same section, is used to evaluate how well the model solves the specific problem relative to some data. In section

1. Deep learning refers to neural networks with many layers that can learn complex details.

4.1, the distinction of supervised- and unsupervised learning will become clear. For supervised learning, the loss function typically compares what the model predicts to a corresponding *label* or *ground truth*. For unsupervised learning, the loss function will be based on other measurements. It is common to use validation- and test data to evaluate how the model performs on unseen data. The validation process is usually done regularly during training to monitor how the model improves and to choose the best model composition, whereas testing is performed on a complete fully trained model. *Hyperparameters* are non-trainable parameters that should be user defined to fit the specific problem.

In this thesis, mostly convolutional neural networks combined with other deep learning methods will be used. CNNs are based on the classical feed-forward neural network, hence this will be introduced at first to get the complete understanding. Other relevant machine learning principles will also be introduced.

4.1 Supervised and Unsupervised Learning

Machine learning is split into two main categories; supervised and unsupervised learning. Supervised learning is when each training sample, $\mathbf{x}^{(i)}$, are provided a label, $\mathbf{y}^{(i)}$. Together, all such samples, $\mathbb{D}_{supervised} = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^N$ where N denotes the cardinality, constitute the training data. The training data enable the supervised machine learning model to learn details in the data that are common for the samples associated to the different labels (Theodoridis and Koutroumbas, 2009). For unsupervised learning, the samples are not provided any label and the machine learning architectures are designed to seek structures in the data without knowing any ground truth about it. The complete training data when doing unsupervised learning is therefore only $\mathbb{D}_{unsupervised} = \{\mathbf{x}^{(i)}\}_{i=1}^N$.

Supervised learning usually produces better results, but needs manually annotated ground truth labels that can be expensive and time consuming to collect. Whether to use supervised or unsupervised learning depends on the task and on available resources. Nowadays, there are methods combining these categories; active learning is one such approach. Active learning asks for human interaction (labeling) on some samples $\mathbf{x}^{(i)}$, and can typically be used for human annotation on difficult data samples. This reduces the amount of human interaction in a labeling process. Active learning is a version of semi-supervised learning, where the complete training data is a combination of labeled and unlabeled data.

4.2 Neural Network

Feed-forward neural networks² (NNs) are the quintessential machine learning and deep learning models (Goodfellow et al., 2016). These are also known as multilayer perceptrons. NNs exist both as supervised and unsupervised, and have been generalized to fit a wide variety of problems. Deep learning models are based on deep neural networks, making it the main contribution to the recent development of machine learning and artificial intelligence (AI).

A neural network aims to fulfill the ideal function $f^* : \mathbf{x} \rightarrow \mathbf{y}$ by using the approximated mapping $\mathbf{y} = f(\mathbf{x}; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ are all parameters in the model, refined to best approximate f^* . The output of the network, \mathbf{y} , depends on the input \mathbf{x} , and all parameters $\boldsymbol{\theta}$ in the layers $\{f^{(l)}\}_{l=1}^L$, where L is the total number of layers. The neural network, described by the mapping $\mathbf{y} = f(\mathbf{x}; \boldsymbol{\theta})$, can then be expressed as in equation 4.1. Here \circ is the Hadamard product, as included in Table 2.1.

$$f(\mathbf{x}; \boldsymbol{\theta}) = \left(f^{(L)} \circ f^{(L-1)} \circ \dots \circ f^{(1)} \right) (\mathbf{x}) \quad (4.1)$$

Each layer $f^{(l)}$ consists of k_l neurons doing individual computations in parallel. By utilizing the neurons in each layer, the continuous mapping in equation 4.1 can be discretized to form a vector expressing the weights of all neurons, and hence a matrix expressing weights between all neurons. Equation 4.2 describes how the scalar output of one neuron in layer $f^{(l)}$ is a result of all neuron outputs in the previous layer $f^{(l-1)}$, and how the *weights* in \mathbf{W} and *biases* in \mathbf{b} affect the computation in the different neurons. A layer on this form is called a *fully connected layer*.

$$f^{(l)} \left(\mathbf{x}^{(l-1)}; \boldsymbol{\theta}^{(l)} \right) = \phi \left(\mathbf{W} \mathbf{x}^{(l-1)} + \mathbf{b} \right), \quad l = 1, \dots, L \quad (4.2)$$

In equation 4.2, the following notation is used:

- $\mathbf{x}^{(l-1)}$ denotes the output of previous layer $f^{(l-1)}$ and $\mathbf{x}^{(0)}$ denotes the input vector.
 - $\boldsymbol{\theta}^{(l)}$ denotes all trainable parameters in the layer $f^{(l)}$, stored in \mathbf{W} and \mathbf{b} .
2. The traditional neural networks are, in their simplest form, often called feed-forward NN because they, after completed training, are intended to pass each training sample $\mathbf{x}^{(i)}$ only in the forward direction of the network without any recursive or feedback loops.

- $\phi(\cdot)$ denotes an activation function operating on vectors, described in detail later in this section.
- $\mathbf{W} \in \mathbb{R}^{k_l \times k_{l-1}}$ denotes the weight matrix for layer $f^{(l)}$, includes trainable parameters. k_l and k_{l-1} denote number of neurons in layer $f^{(l)}$ and $f^{(l-1)}$, respectively.
- $\mathbf{b} \in \mathbb{R}^{k_l}$ denotes the bias vector for layer $f^{(l)}$, composed of trainable parameters.
- $L \in \mathbb{N}$ is the total number of layers in the neural network.

L is referred to as the *depth* of the network. The term "deep learning" originates from deep neural networks, which typically consists of millions of trainable parameters, and hence has the ability to learn complex details in the training data (LeCun et al., 2015).

4.2.1 Activation Function

The activation function is a non-linear function applied on the output of a neuron (as expressed by $\phi(\cdot)$ in equation 4.2). It transforms the linear operation inside a neuron to a non-linear one, making the network able to learn complex details. Activation functions are also usually continuously differentiable, which is important when adjusting the weights. In equation 4.2, the activation function operates element-wise on vectors.

Equation 4.3 (Rottmann, 2003) shows the three dominating activation functions. The sigmoid activation function was habitually mainly used in traditional machine learning architectures. In modern neural networks, the rectified linear unit (ReLU) (Nair and Hinton, 2010) is a default recommendation (Goodfellow et al., 2016). In addition to its simplicity, it has proven to be mathematically optimal. However, the sigmoid activation function is still widely used for single object category classification problems (at the end of the network) as it confines any value to the range $[0, 1]$, and hence make them easy to interpret as pseudo-probabilities. The tanh activation function behaves quite similar to the sigmoid, confining all values to the range $[-1, 1]$. The activation functions in equation 4.3 can be further generalized to adapt specific problems.

$$\begin{aligned}
 \phi_{sigmoid}(x) = \sigma(x) &= \frac{1}{1 + e^{-x}} \\
 \phi_{tanh}(x) = tanh(x) &= \frac{e^x - e^{-x}}{e^x + e^{-x}} \\
 \phi_{ReLU}(x) &= \max\{0, x\}
 \end{aligned} \tag{4.3}$$

4.3 Convolutional Neural Networks

CNNs are definitely one of the most influential result of the work on artificial neural networks. The two-dimensional convolution, as will be introduced in section 4.3.1, makes CNNs perfectly suited for processing of array-like data, such as images. A neural network is called a CNN if at least one of the layers in the net is a so-called *convolutional layer*, as described in detail in section 4.3.4. Architectures used in this thesis are largely based on CNNs, and will therefore be introduced thoroughly before moving over to other machine learning concepts.

4.3.1 Convolution

What defines a CNN, is that, some place in the network, the convolution operation is applied. Hence, this operator is essential to grasp the building blocks of a CNN. The two-dimensional (as used in CNNs when data are images) convolution operation between an input array $X(x, y)$ and a *convolution kernel* $K(x, y) \in \mathbb{R}^{f_h \times f_w}$, in its discrete representation, is defined in equation 4.4. Using a proper kernel, the convolution operation can be used to find specific features in the input array, and is considered as an extremely powerful tool in image processing (Gonzalez and Woods, 2018).

$$(X * K)(x, y) = \sum_{m=x-f_h}^{x+f_h} \sum_{n=y-f_w}^{y+f_w} X(x-m, y-n)K(m, n) \quad (4.4)$$

4.3.2 Pooling

Pooling is an important construct in most CNN architectures as it condenses the information while reshaping the data. In general, a pooling function is a downsampling operation outputting some appropriate statistics of the input. Examples of pooling functions are average pooling and max pooling, where max pooling is widely used in deep CNNs (Goodfellow et al., 2016). An example of average and max pooling of an input array of size 2×2 to a scalar, is described in equation 4.5.

$$\begin{aligned} \text{AveragePool} \left(\begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix} \right) &= \left[\frac{1}{2 \times 2} \sum_{i=1}^2 \sum_{j=1}^2 x_{ij} \right] \\ \text{MaxPool} \left(\begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix} \right) &= \left[\max\{x_{ij}\}_{i,j=1}^2 \right] \end{aligned} \quad (4.5)$$

Usually, masks of a given size (for instance 2×2 , as in equation 4.5) are *strided* over the array, resulting in a downsampled array. Strides apply to both pooling and convolution operations. The stride describes how many steps the convolution kernel or the pooling area is to be moved on the input array between each operation. Pooling is a strong tool in deep learning because it translates the array to a differently shaped representation, while it retains the most informative values and condenses the information.

4.3.3 Transposed Convolution

Transposed convolution (also called fractionally strided convolution, deconvolution or learned upsampling) is a version of the convolution operator mostly used in segmentation networks. While pooling is a suitable operator for down-sampling the image, transposed convolution is used for upsampling. Learned upsampling of images is a strong tool for computer vision, and is a vital part of segmentation models like fully convolutional network (FCN) (Long et al., 2015) and U-net (Ronneberger et al., 2015).

$$\mathbf{X} = (\mathbf{Y} * \mathbf{K})(x, y) \quad (4.6)$$

Equation 4.6 describes a transposed convolution operation³ to produce \mathbf{X} . \mathbf{Y} denotes some array (output from previous layer in the network), \mathbf{K} denotes a convolution kernel, and the convolution operation is described in equation 4.4. By using an appropriate stride, convolution kernel size, and by zero-padding \mathbf{Y} , a learned up-sampling using transposed convolution can be achieved, i.e. $\dim(\mathbf{Y}) < \dim(\mathbf{X})$.

4.3.4 Convolutional Neural Networks

Early in section 4.3, it was stated that a CNN is a neural network where at least one of the layers, $f^{(l)}$, $l \in \{1, \dots, L\}$, is a convolutional layer.

$$f^{(l)}(\mathbf{X}^{(l-1)}; \boldsymbol{\theta}^{(l)}) = \phi(\mathbf{K} * \mathbf{X}^{(l-1)} + \mathbf{b}) \quad (4.7)$$

A convolutional layer can be mathematically described by equation 4.7, where \mathbf{K} is the convolution kernel, $\mathbf{b} \in \mathbb{R}^{k_l}$ is the bias vector, k_l is a hyperparameter denoting the number of convolution kernels in the layer, $\boldsymbol{\theta}^{(l)} = \{\mathbf{K}, \mathbf{b}\}$, and $\mathbf{X}^{(l-1)}$ denotes the output array of the previous layer $f^{(l-1)}$. For layer $l = 1$, $\mathbf{X}^{(l-1)} = \mathbf{X}^{(0)}$ denotes the initial input array. The other terms of equation 4.7 are described in equation 4.4 and 4.2. Dimensions of the different terms

3. Equation 4.6 is initially similar to a standard convolution, but describes transposed convolution if the given conditions are fulfilled.

can be customized to adapt the specific problem. For instance, if initial input array have dimensions $\mathbf{X}^{(0)} \in \mathbb{R}^{h \times w \times c}$, where h is height, w is width, and c is number of channels, the convolution kernel of a layer l will have dimensions $\mathbf{K}^{(l)} \in \mathbb{R}^{f_h \times f_w \times k_{l-1} \times k_l}$. However, dimensions of the kernels and input arrays between layers are obligated to coincide. The output array of the first layer, $\mathbf{X}^{(1)}$, is computed from the convolution kernel $\mathbf{K}^{(1)} \in \mathbb{R}^{(f_h \times f_w \times k_0 \times k_1)}$ and the initial input array $\mathbf{X}^{(0)} \in \mathbb{R}^{(h \times w \times c)}$, and will have dimensions $h \times w \times k_1$.

In section 4.3.2, pooling was introduced and it was stated that operations like these results in downsampled arrays for CNNs. *Receptive field* describes the area of early layers (adjoining input) indirectly represented by deeper layers (distant from input). The receptive field deep in a CNN is dependent on the composition of convolutional layer and pooling layers, and the dimension of convolution kernels and pooling masks. For object detection using deep CNNs, the receptive field may limit the maximum detectable size of objects. This may cause a bottleneck of the model, especially when operating with large scale images ⁴.

Well known benchmark CNNs includes VGG (Simonyan and Zisserman, 2014), AlexNet (Szegedy et al., 2013) and ResNet (He et al., 2016). These networks have a universal architecture which can be adapted to various problems. Such networks report good results, and more complex deep learning models are often built upon these.

4.4 Optimization Problem for Supervised Learning

In section 4.1, the concept of training samples for supervised learning was introduced. The optimization problem for a neural network involves refining all parameters in the network θ , with respect to minimizing a loss function \mathcal{E} , for all training samples $\{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^N$. The optimization problem assumes there exists an optimal solution of the neural network with respect to a loss function for the specific problem. This optimal solution is numerically approximated using optimization methods. The process of optimizing parameters according to some loss function will now be introduced, and state-of-the-art methods for approximating this optimum will be presented.

4. In large-scale images the size of objects varies a lot, it can therefore be challenging to design a model with an appropriate receptive field.

4.4.1 Parameter Optimization

All parameters in the L layers of the neural network $\theta = \{\theta^{(l)}\}_{l=1}^L$ can be refined such that the model best approximates its optimal goal: outputting a value for an input sample $\mathbf{x}^{(i)}$, $\hat{\mathbf{y}}^{(i)} = f(\mathbf{x}^{(i)}; \theta)$, that is equal to the corresponding ground truth label $\mathbf{y}^{(i)} = f^*(\mathbf{x})$ of the sample. To evaluate how well the mapping $f(\mathbf{x}; \theta)$ approximates the ideal mapping $f^*(\mathbf{x})$, a loss function is needed.

A loss function is a function mathematically describing the mapping to be solved in the specific problem. In section 4.1, the concepts of supervised and unsupervised learning were introduced, and the author points to the different data set situations. For unsupervised learning, the loss function evaluates how well $f(\mathbf{x}; \theta)$ approximates $f^*(\mathbf{x})$, without knowing any true labels. For supervised learning, however, which will be focused on here, the loss function is typically a suited dissimilarity measure evaluating the dissimilarity between the networks output $\hat{\mathbf{y}}^{(i)} = f(\mathbf{x}^{(i)}; \theta)$ and the corresponding label $\mathbf{y}^{(i)}$. The loss function will be noted as \mathcal{E} , and the output of the loss function is referred to as loss.

4.4.2 Objective Function

The objective of the optimization process is to (usually) minimize the expected loss under the training data distribution, also called minimizing the objective function.

$$J(\theta) = \mathbb{E}_X[\mathcal{E}] = \mathbb{E}_{X \sim p_{data}(x)}[\mathcal{E}] \quad (4.8)$$

The objective function $J(\theta)$, is the expected loss given all parameters in the mapping θ with respect to the training data, as expressed in equation 4.8. X is a stochastic variable distributed as the training data. To find the expected loss used in the objective function, maximum likelihood estimation is used on all discrete training data points. By minimizing the objective function, all parameters will be optimized θ^* , as expressed in equation 4.9.

$$\theta^* = \arg \min_{\theta} J(\theta) \quad (4.9)$$

The choice of loss function is highly dependent of the problem to solve. The main categories of machine learning problems are regression problems and classification problems.

For regression problems it is common to use a version of the Mean Squared Error (MSE), expressed in equation 4.10, where $\|\cdot\|$ denotes the Euclidean distance, $\hat{\mathbf{y}}^{(i)}$ is the network output, and $\mathbf{y}^{(i)}$ is the true label associated with $\hat{\mathbf{y}}^{(i)}$ (Goodfellow et al., 2016). For classification problems, it is common to use a version of the standard cross-entropy function, expressed in equation 4.11. The cross-entropy loss is closely related to the Kullback-Leibler (KL) divergence, which evaluates the similarity between the data distribution and the current model distribution. When optimizing a model with respect to some data, the data distribution is constant and cannot be optimized, resulting in cross-entropy being an adaption of KL divergence specialized for training classification models.

$$\begin{aligned}\mathcal{E}_{MSE}(\mathbf{y}^{(i)}, \hat{\mathbf{y}}^{(i)}) &= \|\mathbf{y}^{(i)} - \hat{\mathbf{y}}^{(i)}\|^2 \\ J_{MSE}(\theta) &= \mathbb{E} \left[\mathcal{E}_{MSE}(\mathbf{y}^{(i)}, \hat{\mathbf{y}}^{(i)}) \right] \\ &\approx \frac{1}{N} \sum_{i=1}^N \|\mathbf{y}^{(i)} - \hat{\mathbf{y}}^{(i)}\|^2\end{aligned}\tag{4.10}$$

$$\begin{aligned}\mathcal{E}_{cross-entropy}(\mathbf{y}^{(i)}, \hat{\mathbf{y}}^{(i)}) &= -\log \hat{\mathbf{y}}^{(i)} \\ J_{cross-entropy}(\theta) &= \mathbb{E} \left[\mathcal{E}_{cross-entropy}(\mathbf{y}^{(i)}, \hat{\mathbf{y}}^{(i)}) \right] \\ &\approx -\frac{1}{N} \sum_{i=1}^N \left(\mathbf{y}^{(i)} \log \hat{\mathbf{y}}^{(i)} \right)\end{aligned}\tag{4.11}$$

To understand where the $\mathbf{y}^{(i)}$ originates from in the last term of equation 4.11, the reader must recall that $\mathbb{E}[\cdot] = \mathbb{E}_{X \sim p_{data}(x)}[\cdot]$, where $\mathbb{E}[\mathbf{x}^{(i)}] = \mathbf{y}^{(i)}$.

4.5 Optimization Methods

When optimizing the model, we seek a global minimum of equation 4.9. We approximate this by incrementally moving in the gradient direction of the objective function $J(\theta)$, $\nabla_{\theta} J(\theta)$. Numerous algorithms have been proposed for solving this minimization problem in equation 4.9. All such optimization algorithms are based on the traditional gradient descent (Curry, 1944) and stochastic gradient descent (SGD) (Robbins and Monro, 1951). Nowadays, it is common to use the state-of-the-art adaptive moment estimation (Adam), presented by Kingma and Ba (2014). Well known optimization techniques also includes Nesterov accelerated gradient descent (NAG) (Nesterov, 1983), momentum stochastic gradient descent (Qian, 1999) and root mean squared

propagation (RMSprop) (Hinton et al., 2012a).

The gradients of a parameter in the model depend on all other parameters in all layers, and are complex to calculate. In theory, this calculation is feasible for minor algorithms, but not practically desirable for larger algorithms. For these larger algorithms, the backpropagation algorithm is used (Rumelhart et al., 1986). The backpropagation algorithm applies the chain rule for calculating the gradients, starting with layers at the end of network and incrementally moving forward in the network, allowing the weights to be adjusted according to the gradients. In fact, the backpropagation algorithm does not only makes it possible to calculate the gradients, but it is also a highly effective method for refining the weights and training the neural network (Goodfellow et al., 2016)⁵.

4.5.1 Gradient Descent

Gradient descent (Curry, 1944) is a traditional and widely used method for optimizing machine learning problems. For every parameter in the model, θ , with its corresponding gradient, $\nabla\theta$ ⁶, the parameter is incrementally updated according to the rule in equation 4.12 over several *epochs*⁷ until convergence.

$$\theta_{new} = \theta_{old} + \mu \nabla\theta \quad (4.12)$$

In equation 4.12, θ_{new} and θ_{old} denote the parameter values of the current epoch and the previous epoch, respectively, $\mu \in \mathbb{R}$ denotes the *learning rate* (which will be explained later in this section), and $\nabla\theta$ denotes the gradient of θ . At first epoch, all parameters are initialized randomly⁸ or from a pretrained model as presented in section 5.3.1.

Stochastic gradient descent (SGD) (Robbins and Monro, 1951) is a stochastic approximation of gradient descent where the total training data in conjunction, $\mathbb{D} = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^N$, is split into several randomly drawn subsets, $\mathbb{B} \subseteq \mathbb{D}$. The subsets \mathbb{B} , called *batches*, are of size $B \in \{1, 2, \dots, N\} \in \mathbb{N}$, where B is the *batch size*. Each batch is run separately through the network at every epoch, and the batch-elements are typically redefined for every epoch, generating a

5. The mathematically interested reader is encouraged to study the details of the backpropagation algorithm in Theodoridis and Koutroumbas (2009).

6. Where the gradient is typically calculated using the backpropagation algorithm.

7. One epoch is one iteration feeding all training data through the model and updating the parameters once. Several epochs are typically needed to achieve convergence.

8. For deep neural networks, there are typically millions of parameters that needs to be adjusted. For these models to converge, the parameters cannot be unconditionally random initialized. Techniques addressing this problem have been proposed, Glorot and Bengio (2010) and He et al. (2015) presented the techniques for weight initialization which are most used nowadays.

stochasticity in the data. The stochastic gradient descent algorithm is more prevalent and more common than the standard gradient descent. The use of batches provides the opportunity to train models on enormous data sets without draining the workstation capacity. The use of batches do also generalize the model by generating a stochasticity in the data, and is also used in modern optimization algorithms.

Learning rate is a hyperparameter present in most optimization methods. The learning rate restricts the optimizer from doing exaggerated or too moderated steps, and is a hyperparameter typically in the range $\mu \in (0, 1]$. Figure 4.1 shows different situations where the learning rate is erroneous and correctly tuned, and how it affects the optimizer to localize the minimum of the objective function $J(\theta)$. If the learning rate is too low it takes unnecessarily many epochs to localize a minimum. The optimizer may reach a maximum number of epochs before reaching the minimum, or the optimizer risks being stuck in a poor local minimum. If the learning rate is too high, the optimizer may oscillate around the global minimum, preventing convergence.

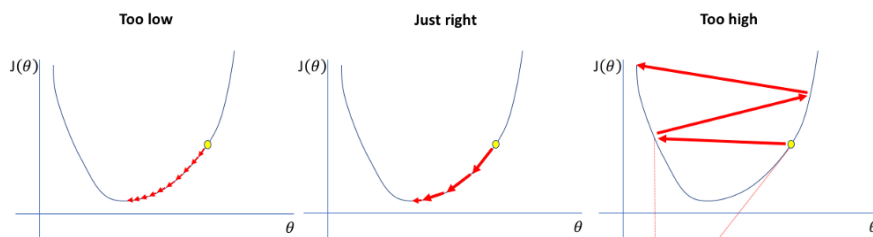


Figure 4.1: Figure visualizing conceptual sketch of how the learning rate affects the optimizer when localizing a minimum of the objective function $J(\theta)$.
Image credits: <https://mc.ai/cyclical-learning-rates-for-training-neural-networks/>. Adopted 11.03.2020

There have been proposed adaptive optimization methods addressing the problem of erroneously tuned learning rate. Adaptive gradient algorithm (AdaGrad) (Duchi et al., 2011) adapts a learning rate for each feature dimension in the optimization space dynamically, while still having a learning rate hyperparameter that gets refined in the separate dimensions. Adadelata (Zeiler, 2012) removes the learning rate hyperparameter completely and, like AdaGrad, it adapts a learning rate to each feature dimension dynamically. These methods are adequate, however, the adaptive moment estimation technique (Adam) utilizes momentum to get a precise and rapid convergence.

4.5.2 Momentum and Adam

There are numerous optimization techniques that include the momentum term. Optimization methods have developed further since the inclusion of momentum in stochastic gradient descent (Momentum SGD) (Qian, 1999), but the main principles of momentum are the same: the optimizer uses the previous step size when evaluating the current step size with a goal of speeding up convergence in the beginning, while doing accurate decisions when converging. In other words, the optimizer does large adjustments when operating far from a minimum, but restricts the step size as it approaches the minimum.

Momentum SGD incrementally updates the parameter θ according to the rules in equation 4.13, where v_{new} and v_{old} are the current and previous momenta, respectively, γ is a hyperparameter, and μ is the learning rate.

$$\begin{aligned}v_{new} &= \gamma v_{old} + \mu \nabla \theta \\ \theta_{new} &= \theta_{old} - v_{new}\end{aligned}\tag{4.13}$$

The Adam (Kingma and Ba, 2014) optimizer extends the concept of momentum, and is considered as the state-of-the-art optimizer for the majority of deep learning problems nowadays. The reason behind Adam's success is firstly that it incorporates momentum directly as an estimate of the first-order moment of the gradient (Goodfellow et al., 2016). Secondly, Adam accounts for inaccuracies in initialization by incorporating bias corrections for both the first- and second-order moment estimates. This corresponds to applying individual learning rates on all parameters, based on first- and second-order moment estimates of the gradients (Hansen, 2019). Algorithm 1 shows a detailed step-wise description of the Adam optimization method where each epoch will update the parameters until it reaches a *stopping criterion*. A stopping criterion can typically be a predefined maximum number of epochs, or when a measure of convergence remains unchanged. In algorithm 1, it emerges that Adam also applies batches, as introduced for the stochastic gradient descent in section 4.5.1, to achieve a stochasticity in the data.

Algorithm 1: Adam optimization method (Goodfellow et al., 2016). Default values suggested by original paper Kingma and Ba (2014) are given in parentheses.

- Initialize Hyperparameters:

Learning rate μ as float ($\mu = 0.001$)

Exponential decay rates for moment estimates, $\rho_1 \in [0, 1)$ and $\rho_2 \in [0, 1)$ as float ($\rho_1 = 0.9$, $\rho_2 = 0.999$)

Numerical stability constant ξ as float ($\xi = 10^{-8}$)

- Initialize:

Parameters θ as floats

1st and 2nd moment variables $\mathbf{s} = \mathbf{o}$, $\mathbf{r} = \mathbf{o}$ as floats

Time step $t = 0$ as int

- Scheme:

while *stop criterion not fulfilled* **do**

 Sample a batch \mathbb{B} of size b from the complete training data set \mathbb{D}

 Estimate current gradient $\mathbf{g} = \frac{1}{b} \nabla_{\theta} \sum_{i=1}^b \mathcal{E}(\mathbf{y}^{(i)}, f(\mathbf{x}^{(i)}; \theta))$

$t = t + 1$

 Update biased 1st moment estimate $\mathbf{s} = \rho_1 \mathbf{s} + (1 - \rho_1) \mathbf{g}$

 Update biased 2nd moment estimate $\mathbf{r} = \rho_2 \mathbf{r} + (1 - \rho_2) \mathbf{g} \odot \mathbf{g}$

 Correct bias in 1st moment $\hat{\mathbf{s}} = \frac{\mathbf{s}}{1 - \rho_1^t}$

 Correct bias in 2nd moment $\hat{\mathbf{r}} = \frac{\mathbf{r}}{1 - \rho_2^t}$

 Compute element-wise update $\Delta \theta = -\mu \hat{\mathbf{s}} \frac{\hat{\mathbf{s}}}{\sqrt{\hat{\mathbf{s}} + \xi}}$

 Apply element-wise update $\theta = \theta + \Delta \theta$

end

4.6 Regularization

A key challenge when it comes to machine learning problems is to assemble architectures that do correct and precise decisions on data they have never seen before, not just on the training data. Strategies explicitly designed to reduce the validation error are called regularization strategies, and numerous techniques can be included in training to achieve a more generalized model⁹. Overfitting of a model is an effect that decreases the generalizability, and batch normalization (Ioffe and Szegedy, 2015), weight normalization, dropout (Hinton et al., 2012b) and employing batches are example of well known regularization strategies that will be presented in the following.

9. Generalizability of a model describes how well the trained model performs on unseen data.

4.6.1 Overfitting

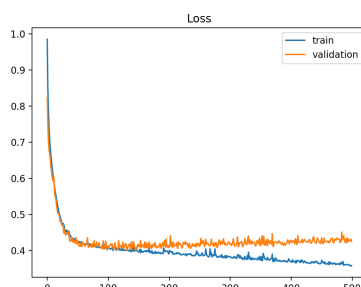


Figure 4.2: Visualization of an example training process with corresponding validation loss for each epoch during training. Here, the performance is evaluated using loss \mathcal{E} for the specific problem. The figure is a conceptual sketch and is not sourced from a model in this thesis.

Image credits: <https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/>. Adopted 12.03.20.

An overfitted machine learning model is producing close to perfect performance scores on the empirical data used to train the model, but is performing significantly poorer on the unseen validation and test data. Figure 4.2 visualizes the overfitting effect through a thought scenario where a model, evaluated by a loss \mathcal{E} defined for the specific problem, is trained over 500 epochs. Without any regularizing constraints, the optimizer does its best to minimize the training loss throughout the training process. After exceeding ~ 100 epochs, the reader may notice that the training and validation losses start to diverge. The model is now "learning the data set", not the distribution, and the validation loss is starting to increase. Regularization strategies preventing this effect will now be presented.

4.6.2 Weight Regularization

Weight regularization restricts the model parameters θ in the deep learning model from becoming independently too great in value, and it penalizes nonzero parameters (Goodfellow et al., 2016). An exaggerated large parameter value is typically indicating that the model is learning the data set (not the data distribution), resulting in poor generalizability, and can be limited by suppressing such values. By penalizing nonzero parameters, sparse and structural networks will be encouraged. In practice, weight regularization is commonly achieved by including the L_1 norm, penalizing nonzero parameters, and the L_2 norm, penalizing large parameter values, in the objective function $J(\theta)$ (Wen et al., 2016).

4.6.3 Batch Normalization

For deep neural networks, composed of numerous fully connected and convolutional layers, small parameter updates in one layer will affect the input to subsequent layers (in feedforward networks) (Goodfellow et al., 2016). This effect is called *internal covariate shift*, and is said to delay the optimization. Ioffe and Szegedy (2015) proposed the batch normalization (often shortened as batch norm) strategy for suppressing all features in a batch, while retaining the structure, to achieve smaller adjustments in the weights. Traditionally, batch normalization has been considered a technique to avoid doing unnecessary many updates for a considerable amount of parameters before convergence, and hence avoid the challenge of internal covariate shift. However, Santurkar et al. (2018) concluded that batch normalization reparametrizes the underlying optimization problem to make it more stable, rather than directly avoiding the internal covariate shift. Either way, batch normalization achieves increased optimization. It is achieved by applying equation 4.14 to each layers input, where $\hat{\mathbf{x}}^{(i)}$ is the batch normalized feature vector of $\mathbf{x}^{(i)}$ and $\mathbb{E}_{\mathbb{B}}[\cdot]$ and $Var_{\mathbb{B}}[\cdot]$ denote the expectation and variance with respect to the set \mathbb{B} .

$$\hat{\mathbf{x}}^{(i)} = \frac{\mathbf{x}^{(i)} - \mathbb{E}_{\mathbb{B}}[\mathbf{x}^{(i)}]}{\sqrt{Var_{\mathbb{B}}[\mathbf{x}^{(i)}]}} \quad (4.14)$$

In section 4.5, the reader was introduced to the concept of batches, which are used when batch norm is practiced. In fact, the use of batches itself acts as a regularization strategy because the presented data vary in each iteration ¹⁰. Optimization using batches acts as an approximation to optimization using the complete distribution (the complete training data) and introduces a stochasticity in the data that presumably increases the generalization properties. By using batches and batch normalization, the need for other regularization strategies can be considered as eliminated (Redmon and Farhadi, 2017).

4.6.4 Dropout

Dropout (Hinton et al., 2012b) is a technique with several motivations. The most prevalent motivation is that dropout is used as a regularization strategy to avoid overfitting during training. Figure 4.3 concretises the idea of applying the dropout strategy as part of the training process. By randomly dropping units (typically neurons), with a probability p , a typical situation where a unit is only learned to operate in the context of specific parallel units can be avoided. The objective is units that are learned to operate more independently. During test time, all units are used.

10. Iteration here refers to each time a batch is presented to the model.

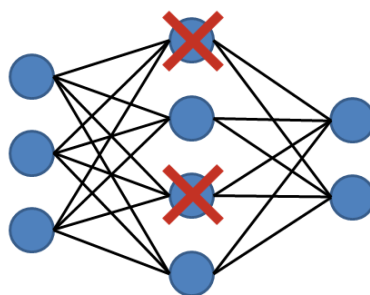


Figure 4.3: Conceptual sketch of how the dropout strategy, with $p = 0.5$, applies to a neural network. Image credits: <https://towardsdatascience.com/coding-neural-network-dropout-3095632d25ce>. Adopted 13.03.20.

4.6.5 Data Augmentation

To increase the generalization properties of a model, it is advantageous to train it on all different scenarios that can be faced during test time. This can be done by artificially generating data related to the originally observed data. For images, this is typically done by flipping vertically, flipping horizontally, rotating, scaling, or by adding noise (Goodfellow et al., 2016). However, it is not beneficial to train the model on augmentations that are not physically valid, especially if they cannot even be argued to be pragmatically valid (in the sense of not being advantageous in the learning process) (Brenn et al., 2019). The concept of data augmentation to artificially increase the amount of training data is especially relevant to data sets of limited size (Cheng et al., 2016). The discussion of augmenting satellite images continues in section 7.3.

4.7 Performance Measures

To evaluate how well the model performs, there is need for an appropriate measure. In section 4.4, it was stated that a loss function is a measure of how well the mapping $f(\mathbf{x}; \theta)$ approximates the ideal mapping $f^*(\mathbf{x})$, i.e. how well a predicted value coincides with the corresponding true label for that specific problem. Two common loss functions were also explicitly presented. A loss function is a great performance measure for a model with its corresponding data, but can not be generalized if one for instance want to test the data on a completely different model. Also, the loss itself is difficult to interpret. Common performance measures, which are easier to interpret and generalize, will be introduced in the following. Which measure to use depends on the situation, but common for supervised learning is to, in some way, compare the output of the model with prior information on corresponding ground truth label. Such performance measures are typically applied during validation, and will be denoted as \mathcal{A} .

4.7.1 Common Measures

The performance measures that will be introduced here are used for different machine learning problems, not only for object detection architectures used in this thesis. Common for these measures is that they are only suitable for supervised problems, assumed to provide a label.

4.7.1.1 Accuracy

For categorical machine learning problems (classification), a common superior measure is the *overall accuracy* (OA) evaluating how large proportion of data points that were correctly classified. However, for imbalanced classes where one class is highly underrepresented, the OA would still report acceptable results by just ignoring this underrepresented class. In such cases, it is convenient to inspect the *class-conditional accuracy*.

For categorical problems with a moderate number of classes, an orderly way of presenting data is by using a *confusion matrix*. A typical two-class classification problem is to predict a data sample to be positive or negative. This thesis regards evaluating regions as ship contra not ship, and it is often convenient to operate with positive/negative classification in such cases. Soon, the reader will be introduced to how this can be applied to region-based predictions (as is can be applied to most object detection settings). This leads us onto the four terms: true positive (tp), false positive (fp), true negative (tn) and false negative (fn), to evaluate whether a prediction is correct or incorrect with respect to its label. These terms are the ones to be presented in a confusion matrix. In the scientific field of ship detection, these terms are assigned some names: *correct prediction* is a tp, *false alarm* is a fp, and *miss* is a fn. tn is not assigned any particular name.

4.7.1.2 Precision

The performance measure *precision* is a direct result of the recently introduced terms in a two-class confusion matrix: tp, fp, tn, fn. Precision is the proportion of all predicted positives that were correctly classified, and is an indicator of the amount of false alarms. The general precision is defined in equation 4.15.

$$\mathcal{A}_{Precision} = \frac{tp}{tp + fp} \quad (4.15)$$

For object detection, average precision (AP) is a popular measure for evaluating its performance.

AP is a per-class measure taking the average precision with respect to all object categories individually. For the positive/negative classification approach used in this thesis, there is only one AP; for the ship class. In such single object category problems, the AP is reduced to precision, $\mathcal{A}_{AP} = \mathcal{A}_{Precision}$. For multi object category problems, \mathcal{A}_{AP} is calculated per class. Mean average precision (mAP) denotes the average \mathcal{A}_{AP} for multi object category problems, and is an indicator of overall performance.

4.7.1.3 Intersection-over-Union

To determine if a region-based prediction is corresponding to a ground truth region, there is need for an additional measure. Here, the intersection-over-union (IoU) comes in handy. A predicted object is typically concluded to be a tp if $IoU(\hat{\mathbf{y}}^{(i)}, \mathbf{y}^{(i)}) > \tau_{IoU}$, where τ_{IoU} is a user defined threshold.

IoU is a measure of how much two regions overlap relative to their size, and is a common measure for goodness of fit of a predicted bounding box with respect to some ground truth bounding box. The IoU between two bounding boxes Ψ and Γ is defined in equation 4.16.

$$IoU(\Psi, \Gamma) = \frac{area(\Psi \cap \Gamma)}{area(\Psi \cup \Gamma)} \quad (4.16)$$

/5

Object Detection using Deep Learning

Object detection is a vital part of the computer vision research field. The state-of-the-art in automated object detection models is constantly developing. Machine learning models, and specifically CNNs, have shown exquisite performance and is the foundation of most object detection architectures nowadays. In this thesis, the term object detection refers to object detection in images.

Object detection aims to recognize structures and shapes, and process this information to, in some way, describe objects in an image. This includes reporting present objects, annotating them using bounding boxes, or labeling each pixel with the identity of the category it belongs to (Goodfellow et al., 2016). In this thesis, the aim is to apply object detection using machine learning principles to detect ships in satellite images. For ship detection using machine learning, there are two architecture paradigms: models based on semantic segmentation¹ (from now on referred to as segmentation-based models) and models based on bounding boxes² (often called object detection models).

Segmentation-based models will output a two-dimensional array proportional

1. Semantic segmentation models label each pixel in the image with the identity of the object it belongs to.
2. Models based on bounding boxes annotate objects with boxes that enclose the object.

to the size of the original image, where the value of each pixel represents the predicted categorical label at this pixel. In the field of ship detection, segmentation-based models are traditionally essentially used on SAR, images where the spatial resolution is limited. Segmentation-based models will be studied further in section 5.1, and later in section 6.2.

Bounding box-based models typically output a bounding box enclosing the object, but also classification and score for all objects in the image. Bounding box models are again typically divided into proposal-based and grid-based models, where a consideration of speed versus accuracy is important for the choice of model. With the constant development of remote sensing technology, bounding box-based models are well suited for optical satellite images with their exquisite spatial resolution. Bounding box-based models are the main scope of this thesis and will be studied further in section 5.2, and state-of-the-art models will be presented in section 6.3.

Single object category detection versus multiple object category detection is another discrimination for object detection that applies to both bounding box- and segmentation-based models³. Single object category detection is suited for answering specific problems (such as detecting ships), while multiple object category detection are useful for a universal and complete image understanding. A typical image appealing to single object category detection have objects that are small and distributed inhomogeneously and sparsely over the image. This is most commonly used for commercial applications⁴. A typical image appealing to multiple object category detection consists of everyday objects in natural scenes that are large and covers the image densely. This is most commonly used in academic research⁴.

For supervised learning, the *level of precision* of the provided labels is an important factor when designing the model. The level of precision for labels specifies how accurate the objective is described by the label (Han et al., 2014). Lowest level of precision for object detection is typically an annotation describing the number of a specific object in the image, for instance: "There are at least one ship in this image". Approaching higher levels of precision for object detection, labels constituting bounding boxes enclosing all objects in the image can be found. At the very highest level of precision for object detection, a pixel-wise description of represented class can be found (as preferably used in segmentation-based models). Techniques aiming at achieving results of a higher level of precision than the provided labels are constantly proposed.

3. The number of categories (classes) to detect is just a fixed preference and do not affect the model design to such a great extent.

4. Claimed in the Royer and Lampert (2020) presentation notes: <https://cvml.ist.ac.at/talks/lampert-cdem12020.pdf>. Adopted 17.03.20.

Weakly supervised learning is the main category of such techniques (Brenn et al., 2019).

This thesis will mainly focus on bounding box-based models. However, segmentation-based models are essential in object detection using machine learning, and will be partly included for a complete perspective.

5.1 Segmentation-based Models

As stated earlier in this chapter, segmentation-based models for ship detection are mostly used when it comes to SAR images. For optical remote sensing images, both bounding box-based and segmentation-based models are widely used. Segmentation-based models, operating with pixel-wise classification, use labels of highest level of precision and are generally computationally expensive.

Common for segmentation-based deep learning architectures is a complete convolutional architecture, i.e. no fully connected layers. The input image is interpreted and downsampled several times before repeated upsampling using transposed convolution and a comparison with a label map of identical shape as the input image at output. In addition to this standard design, different modifications are typically included for increased performance on specific problems.

For large scale images with underrepresented objects, which is the situation for ship detection using remote sensing, a model can report satisfactory performance results by just ignoring the objects. In such situations, a loss factor compensating the extreme class imbalance is required. Focal loss (Lin et al., 2017c) is one successful and widely used approach addressing the class imbalance problem. FCN (Long et al., 2015) and U-net (Ronneberger et al., 2015) are considered benchmark and state-of-the-art models for general semantic segmentation, and forms the foundation of most segmentation-based object detection models⁵.

5. Segmentation-based models are mentioned to get a complete perspective of object detection using remote sensing, but a thorough presentation is omitted due to time limitations, as it is outside the scope of this thesis.

5.2 Bounding Box-based Models

Where segmentation-based models describe an image and its objects in a pixel-wise manner, bounding box-based models describe all objects using bounding boxes (BBoxes) θ_{BBox} , and non-described areas are evaluated as "background". The predicted BBoxes are typically the output of the last layer in the detector. Each BBox is typically parameterized by $\theta_{BBox} = \{\theta_x, \theta_y, \theta_w, \theta_h, c\}$, where $\theta_x, \theta_y, \theta_w, \theta_h$ are the center x -coordinate, center y -coordinate, width and height of the BBox, respectively. We denote c as the confidence score of single object category problems, which can easily be expanded to describe multiple object category problems.

Bounding box-based object detection architectures are typically sorted into the two paradigms: proposal-based and grid-based models. What characterizes proposal-based object detection models is the pipeline where a selection of regions to contain objects are first proposed, after which these regions are sent into another network for validation and prediction. Grid-based models are often called single-shot models, as the image only goes through a single forward pass during inference. What characterizes grid-based models is the predefined grid uses to split the image into smaller regions to be evaluated separately. Proposal-based models are known to be precise, but computer-intensive and slow, whereas grid-based models are somewhat more imprecise, but are fast and have potential for real-time processing during inference. Models that are a hybrid of proposal- and grid-based principles have also been proposed with a goal of achieving models that are both fast and precise. Royer and Lampert (2020) is a version of a such a hybrid model.

5.2.1 Non-maximum Suppression

Common for proposal-based and grid-based object detection models is that they apply the concept of non-maximum suppression (NMS) just before the predictions are presented to the user. Bounding box-based models will typically have numerous proposed bounding boxes for each object, where only one of them encloses the object in a best possible manner with respect to correct classification on pixel level. There are different methods for performing NMS, but the principle is that, as the name so elegantly describes, all predicted bounding boxes that are not the best predicted bounding box of that object, are removed.

Usually, NMS is performed by first sorting all predictions according to the confidence score. Next step is to, starting at prediction with highest confidence score, calculate the IoU between the current prediction and all predictions with higher confidence score. Only predictions where this IoU is below a user defined threshold and the confidence score is above a user defined threshold will be accepted.

5.2.2 Proposal-based Methods

As introduced early in this section, the main characteristics of proposal-based object detection models is the pipeline where a selection of regions that potentially contain objects are first proposed (called *region proposals*, *anchors* or *prior bounding boxes*), then these regions are sent further into the network for prediction and validation of existence. The region proposal part will be referred to as phase one of the pipeline, and the prediction and validation of existence part will be referred to as phase two.

A wide variety of techniques for proposing anchors have been suggested, and many architectures refine and regress on the first proposed anchors during the second phase. By applying regression in the second phase, the amount of proposed regions in the first phase can be minimized.

5.2.2.1 Region Proposal Techniques

Generic proposal-based object detection architectures have moved from previously using dense sliding window approaches for region proposal, to now using sparser region proposal frameworks, typically a trained region proposal network (RPN) (Kong et al., 2016).

Sliding window approaches will in principle propose anchors at predefined positions with predefined size and aspect ratios. For such approaches, the anchors will rarely fit objects perfectly, and the amount of proposed regions are massive. Most of these proposed regions are not accepted as objects, resulting in unnecessary computer intensive models.

Selective search (Uijlings et al., 2013) is a benchmark region proposal algorithm which is widely used in object detection. It performs image segmentation to achieve a universal image understanding, and proposes anchors based on this. Selective search extracts $\sim 2,000$ region proposals, which is a lot less than the typical sliding window approach (Kong et al., 2016). Also, the anchors are now arbitrarily shaped and positioned, leading to a more precise enclosing of objects. Selective search is the origin behind the success of the pioneering proposal-based object detector called region-CNN, commonly known as R-CNN (Girshick et al., 2014).

State-of-the-art proposal-based object detectors nowadays typically consists of trainable RPNs, leading to end-to-end trainable architectures ⁶. RPNs are the designator for trainable networks, and they can be implemented in different forms. Another pioneering proposal-based object detector applying a RPN for

6. In an end-to-end trainable architecture, all components in the model can be learned using machine learning. This is generally preferred, as it guides every component to do precise and contextual decisions w.r.t. the other components.

region proposal is the Faster R-CNN model (Ren et al., 2015). In this network, the RPN is built in a fully convolutional structure to propose anchors that are passed on to phase two of the architecture. Unlike selective search (Uijlings et al., 2013) and R-CNN (Girshick et al., 2014), this network is end-to-end trainable, which enables the network to learn which proposed regions that were accepted as objects at the end of phase two. This results in fewer and more deliberated region proposals.

5.2.2.2 Region of Interest Pooling

To understand the architectures that will be introduced and described in the following, an introduction of Region of Interest (RoI) pooling is expedient. RoI pooling appears as a layer in modern proposal-based object detection networks, and is supportive to designate a specific region of interest in a large feature map of a fixed size (Girshick, 2015). Neural networks and their weights depend on fixed sized inputs and outputs making it expedient to combine fixed-size feature maps with RoI pooling instead of modifying the data shapes.

Recall the concept of pooling and stride from section 4.3.2. The pooling operation has the ability to retain the key details (RoI) of an array while altering the shapes. A RoI pooling layer is therefore used to transform an arbitrary shaped array to a fixed shape array (or feature vector) while maintaining the key details. For object detection networks, these arbitrary shaped arrays refers to region proposals. The technique for transforming an arbitrary shaped array to a fixed shaped array is to define the pooling stride according to the two arrays.

5.2.2.3 Benchmark Proposal-based Object Detection Models

A conceptual description of benchmark proposal-based object detection models will now be given, before models concerning the ship detection task are described in detail in section 6.3.1. R-CNN (Girshick et al., 2014), Fast R-CNN (Girshick, 2015) and Faster R-CNN (Ren et al., 2015) are considered benchmark models within proposal-based object detection, where Faster R-CNN is still considered as state-of-the-art for general proposal-based object detection ⁷.

Figure 5.1 visualizes the pipeline and the main components of the pioneer proposal-based architecture, R-CNN (Girshick et al., 2014). As mentioned, R-CNN uses selective search for region proposal in phase one. Phase two consists of feature extraction on each prior BBox using a CNN, and a support vector machine (SVM)⁸ to classify the presence of an object within the proposed

7. An excellent summarizing review is given at: <https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e>.

8. SVM is a traditional, strong and flexible tool for classification.

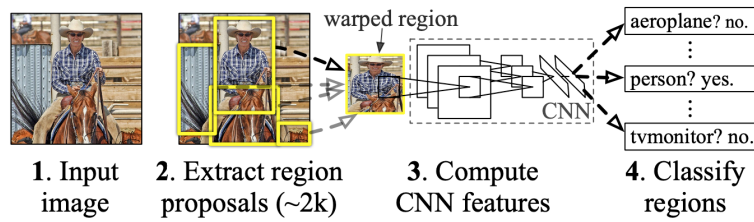


Figure 5.1: R-CNN pipeline with its main components. Image credits: Girshick et al. (2014).

region based on the extracted features. In addition to the SVM, the extracted features are also used to predict offset values of the proposed region, which refines the prior bounding box to increase precision. In object detection, this refinement is often called *regression* as it adjusts the original parameters with the aim of perfectly coinciding with ground truth boxes.

Fast R-CNN (Girshick, 2015), presented by the same author as R-CNN, is a continuation of R-CNN addressing some of its drawbacks. Instead of feeding $\sim 2,000$ region proposals to the CNN, the image itself is input to the CNN resulting in a *convolutional feature map*. A convolutional feature map is a different representation of the input data, which often holds useful information on contours and shapes. Selective search is applied to the convolutional feature map to generate region proposals. These region proposals are again processed by a RoI pooling layer to achieve a fixed shaped representation. Finally, these results are used for classification and regression on each proposed region, similarly as in R-CNN.

Faster R-CNN (Ren et al., 2015) takes a step further from selective search and adopts a trainable RPN that can learn how a prior bounding box should be formed to be accepted. Similarly as for Fast R-CNN, the image itself is input to a CNN to generate a convolutional feature map. The RPN processes the convolutional feature map to predict region proposals. The region proposals are, again, reshaped to a fixed shape using a RoI pooling layer, before classifying and regressing the individual region proposals at the end.

Table 5.1 presents a comparison of speed and precision of the presented benchmark proposal-based object detection models. The reader presumably notices that Faster R-CNN is superior among the proposal-based models.

Table 5.1: A comparison of speed and mAP of the five benchmark models: R-CNN, Fast R-CNN, Faster R-CNN, YOLO and SSD. Results are from an experiment performed on PASCAL VOC 2007 data set (Everingham et al., 2010). Sources: <https://dzone.com/articles/from-r-cnn-to-faster-r-cnn-the-evolution-of-object> and <https://towardsdatascience.com/review-ssd-single-shot-detector-object-detection-851a94607d11>. Adopted 18.03.20.

	R-CNN	Fast R-CNN	Faster R-CNN	YOLO	SSD
Inference time per image [s]	50	2	0.14	0.02	0.017
mAP	66.0	66.9	69.9	63.4	68.0

5.2.3 Grid-based Methods

Where proposal-based models aim to localize independent objects using regions, grid-based models aim to get a complete and universal image understanding of all objects in the image by using a predefined grid. In the beginning of chapter 5, it was stated that multiple object category detection usually is used to get a universal image understanding, and that a typical image for such a scenario consists of everyday objects in natural scenes that are large and cover the image densely. Traditionally, grid-based models are better for predicting such large objects, while proposal-based models are better to detect smaller details.

Grid-based object detectors are efficient and can often operate in real-time, but objects need to be distributed homogeneously on the predefined grid for the model to report satisfactory results (Royer and Lampert, 2020). Figure 5.2 visualizes the main components of the state-of-the-art grid-based object detection model You only look once (YOLO) (Redmon et al., 2016). Common for all grid-based model is the predefined grid, which is applied to the input image to divide it into smaller regions. This grid is user-defined and should match the specific problem under consideration.

A frequent approach is to choose the grid size according to a worst case scenario for the processed images (Royer and Lampert, 2020). To ensure no objects are undetected, even for regions with high object density, the grid cell size should be adequate. A small grid cell size is also a drawback in the sense that the number of operations scales quadratically ($\mathcal{O}(n^2)$) with the number of regions, and most cells will remain empty in the typical object detection problem.

Different architectures for grid-based object detection have been proposed. Common for them all is that they achieve an impressive speed by processing each image only once, however, this may cause smaller objects to remain undetected.

5.2.3.1 Benchmark Grid-based Object Detection Models

A summarizing conceptual description of benchmark grid-based object detection models will now be given, before models concerning the ship detection task are described in detail in section 6.3.2. The single shot detector (SSD) (Liu et al., 2016a) and YOLO (Redmon et al., 2016) algorithms are considered benchmark models within grid-based object detection and were some of the first successful models of this kind. Numerous extensions of these models have been proposed, but the original models are still powerful and form the foundation of grid-based object detection.

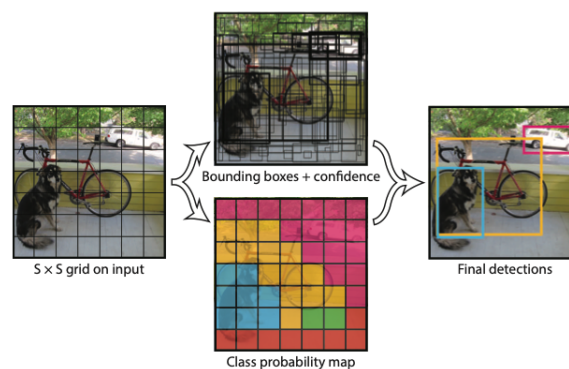


Figure 5.2: YOLO architecture with its main components. Image credits: Redmon et al. (2016).

The design of YOLO (Redmon et al., 2016) is visualized in Figure 5.2. One single CNN is used to predict both the bounding boxes and the class probabilities for these boxes, which is the main reason behind its speed. YOLO first splits the image into a grid of size $C \times C$, resulting in C^2 cells. Based on this grid, BBoxes are defined as a compositions of cells⁹, as visualized in the middle top figure in 5.2. For each BBox, the CNN predicts a classification score together with some offset values. The offset values are regressed parameters adjusting the BBox location to better fit the object. NMS are combined with an acceptance threshold for the classification score to evaluate which of the BBoxes to accept.

SSD (Liu et al., 2016a) follows the concept briefly presented in Figure 5.3. The image is first presented to a CNN, used as feature extraction, to get different representation of where the objects are localized. The BBox size and aspect ratio, as seen in Figure 5.3, subfigure (b) and (c), are predefined by the user to fit the specific problem. BBox locations are evaluated from the feature extraction results, and are learned using end-to-end training. All shapes and aspect ratios at all locations are used to predict classification scores and offset values to better fit the object. Similarly as for YOLO, NMS and an acceptance

9. How these BBoxes should be formed is learned during training.

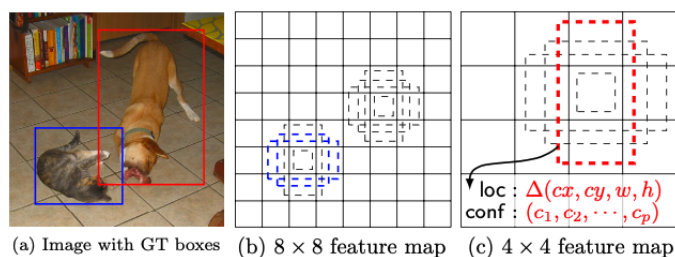


Figure 5.3: SSD framework. Subfigure (a) displays an arbitrary input image with ground truth BBoxes. Subfigures (b) and (c) visualize some default BBox with different sizes and aspect ratios before evaluation, and how location offsets and class scores are predicted.

Image credits: Liu et al. (2016a).

threshold for the classification score is applied to conclude which BBoxes to accept.

Results of speed versus mAP for YOLO and SSD are also included in the comparison of benchmark object detection models in Table 5.1. These results show that grid-based models report promising results. The mAP is fairly good compared to the slower proposal-based models. YOLO and SSD can operate in exceptional 45 frames per second (fps) and 59 fps, respectively, making them well suited for real-time inference problems. These results are reported from an experiment on the PASCAL VOC 2007 dataset (Everingham et al., 2010), which mostly contains large objects covering the images densely. In the field of ship detection in remote sensing images, there may be a risk that smaller ships will not be detected. A study of errors in YOLO shows that it makes a significant amount of localization errors compared to state-of-the-art proposal-based object detectors (Redmon and Farhadi, 2017).

5.2.4 Rotation Invariant Networks

Common for EO images is that objects will be arbitrarily oriented. It is therefore desirable to modify deep CNN for object detection to process the images correctly and do valuable predictions for all object orientations. Naively, the convolutional filters in the network will precisely learn the object characteristics, including the rotation. For remote sensing images, having arbitrarily oriented objects, we want our network to be invariant to these rotations. A rotation invariant network is a network having a mapping $f(\mathbf{x}; \boldsymbol{\theta})$ that is independent on the rotation of the data (Theodoridis and Koutroumbas, 2009). A CNN is translation invariant from its architecture design. A common approach to synthesize rotation invariance in object detection networks is to train it on a wide variety of object rotations. This can be called rotation pseudo-invariance. In a rotation pseudo-invariant network, the network will learn that different object rotations may occur. To achieve this, it is necessary to have enough

training data or to augment the data properly¹⁰. Rotation invariance concerns the loss and decisions of a neural network, but has nothing to do with the label appearance. This is where rotatable bounding boxes (RBox) (often called oriented bounding box (OBB)) come in handy.

5.2.5 Rotatable Bounding Boxes

RBox is an extension of the traditional BBox. For the traditional BBox, the size and aspect ratio may not reflect the real shape of the object, and a lot of background pixels could fall into the box (Liu et al., 2017a). This will create an uncertainty when classifying objects inside the BBoxes. In Figure 5.4, the subfigure on the left hand side also visualizes the problem of using traditional BBoxes on neighboring objects.

The RBox introduces one additional parameter on top of the five existing parameters for a BBox: $\theta_{RBox} = \{\theta_x, \theta_y, \theta_w, \theta_l, \theta_\phi, c\}$. Here θ_ϕ is a trainable parameter representing the orientation of the BBox. The right hand side subfigure in 5.4 visualizes how much background that can be omitted and how great an improvement there is when adopting RBoxes. There are also some drawbacks when using RBox over BBox. One additional parameter typically leads to longer training- and test time, and there are in principle need for many more region proposals for proposal-based models.



Figure 5.4: Image tile visualizing how the RBox is able to locate the ship more precisely, while excluding background from the bounding box, as compared to the traditional BBox. Image is from the Airbus ship detection data set (Kaggle, 2018).

In the field of ship detection, most detectors do not distinguish between bow and stern of a ship. In such cases, if the predicted RBox has an angle θ_ϕ , the true ship heading will lie in the set $\{\theta_\phi, \theta_\phi \pm 180^\circ\}$.

10. Data augmentation was introduced in section 4.6.5 and will be studied further in section 7.3.

5.3 Deep Learning Computation

With the growing popularity of deep learning over the later years, several deep learning frameworks have been implemented and published as open source (Bahrampour et al., 2015). They are also a lot of the reason behind the efficient development in deep learning. Deep learning frameworks are often user-friendly and allow anyone to experiment on machine learning. They are usually computationally faster than self-defined layers, gradients and optimizers, and relax the requirement of heavy mathematical insight. Deep learning frameworks numerically approximate gradients using automatic differentiation (Paszke et al., 2017). The inclined reader is encouraged to study the concept of automatic differentiation.

Available deep learning frameworks includes, but are not limited to, Caffe (Jia et al., 2014), Pytorch (Paszke et al., 2019), Fastai (Howard and Gugger, 2020), Tensorflow (Abadi et al., 2015), and Keras (Chollet et al., 2015). Keras is a high-level framework running Tensorflow back-end. Fastai is a high-level framework running Pytorch back-end. Many modern frameworks have also implemented support for initializing networks using pretrained weights.

5.3.1 Pretrained Deep Learning Models

When training a deep neural network, millions of parameters are fine-tuned to learn the necessary information. Generally, these parameters are randomly initialized before training. If the amount of training data is not of significant size, the parameters will not manage to learn the main contours and crucial characteristics which is needed to make reasonable predictions.

A common solution is to initialize the network with already trained weights of a similar problem, instead of a random initialization. If an architecture includes a widely used neural network, this part of the architecture can be initialized using pretrained weights. For instance, a lot of deep learning models include well known CNNs such as VGG (Simonyan and Zisserman, 2014), ResNet (He et al., 2016) and AlexNet (Szegedy et al., 2013) somewhere in the architecture. There are published open-source weights for these models that are pretrained, often on ImageNet (Russakovsky et al., 2015). If a ship detector is initialized in such a manner, it can directly focus on the characteristics of a ship, instead of using time to learn basic concepts that are similar for all images.

/6

Object Detection Neural Networks and Ship Detection

In chapter 5, the reader was introduced to different paradigms of object detection using machine learning. A review on ship detection in remote sensing images reveals that for SAR images, segmentation-based models are mainly used to obtain a pixel-wise prediction of ships. For optical satellite images, with their high spatial resolution, bounding box-based models are mostly used.

In this chapter, the reader will be given a presentation of the object detection literature, with a particular focus on the ship detection task. Faster R-CNN (Ren et al., 2015), DRBox (Liu et al., 2017a), YOLOv2 (Redmon and Farhadi, 2017), Tiny YOLO and ODGI (Royer and Lampert, 2020) will be particularly reviewed, as these are considered the core of the thesis, and will be experimented on in part IV as well. Traditional ship detection methods are introduced to give an impression of how deep learning revolutionized the research field of ship detection. Segmentation-based ship detection is reviewed for an overall impression of how such models play a significant part of ship detection in remote sensing images. State-of-the-art bounding box-based models that are not specifically studied in this thesis are also included to give an updated status on the research field.

6.1 Traditional Ship Detection

Traditionally, ship detection in remote sensing images is based on analyzing the statistics of the scene, and use detection theoretical techniques upon this. The most common ship detection methods are based on a constant-false alarm rate (CFAR) detector¹ assuming that the background pixel intensities (non-ship pixels) follow a known distribution, for instance a gamma distribution (Eldhuset, 1996). The principle of CFAR is to adaptively determine a detection threshold while preserving a defined number of false alarms (Tao et al., 2016), which is possible for algorithms that satisfy the so-called CFAR property. A CFAR detector therefore models the statistical pixel distribution, while utilizing formal hypotheses with a probabilistic interpretation. However, these models are based on processing of single-pixels, resulting in non-contextual decision making which makes these detectors less suitable for the high spatial resolutional instruments found in modern satellites and images with distributed targets that extend over many pixels.

Recently, deep learning based ship detection architectures have shown state-of-the-art performance. Most object detection models nowadays use deep learning to learn different complex and non-complex scenes. Before embarking the state-of-the-art deep learning architectures, the reader will get a brief introduction to traditional machine learning principles that ruled object detection in remote sensing images the first decade of this millennium.

Cheng and Han (2016) present an excellent overview of the scientific journey in the field of object detection in remote sensing images, with a focus on machine learning techniques. *Template matching* defines an ideal shape of the object to find, in this case; a ship, and a similarity measure is used to evaluate the goodness of fit according to shape, color and other features. Template matching in remote sensing images is tested in for example Liu et al. (2013).

Other strong machine learning architectures for object detection in remote sensing images typically consists of bringing the data into a strong feature representation, then apply a trained classifier to find the features of interest. Such classifiers include the support vector machine (SVM) (which has been used in Han et al. (2014), Xu et al. (2009) and Zhang et al. (2014)), AdaBoost (which has been used in Grabner et al. (2008)) and conditional random field (as has been proposed in Zhong and Wang (2007)). To get a suitable feature representation, the bag-of-words (BoW) model (Li and Perona, 2005) is a strong and popular tool. Another widely used tool for transforming data into a suitable feature representation is the histogram of oriented gradients (HOG) (Dalal and Triggs, 2005). The BoW model is very simple, yet efficient and also invariant to

1. Recall from subsection 4.7.1 that false alarms are reported detections not coinciding with a ground truth ship.

viewpoint variation (Cheng et al., 2016). The HOG is somewhat more complex, but has been widely acknowledged as one of the best models for capturing edge and shape of objects, and is considered as an even stronger tool for feature representation than BoW.

Ship detection using deep learning has now largely taken over. This brings us over to object detection using segmentation-based and bounding box-based models.

6.2 Architectures Based on Segmentation

As stated earlier, segmentation-based models are not directly studied and experimented on in this thesis, but are briefly included for a holistic perspective because segmentation-based detectors are widely used in the field of ship detection. Most such object detectors are based on the standard FCN (Long et al., 2015) and U-Net (Ronneberger et al., 2015) architectures. An example model where FCN has been further developed to generate more reliable predictions is the Res-FCN (Lin et al., 2017a).

6.2.0.1 Res-FCN: Lin et al. (2017a)

² Resnet-50 modified into a fully convolutional network (Res-FCN) combines a fully convolutional structure and a multidimensional attention network to reliably predict ships in remote sensing images. The attention network is trained to localize bow and stern of all ships. These locations are then used to predict the total shape of the corresponding ships, with the objective of doing better predictions on adjacent ships.

6.3 Architectures Based on Bounding Boxes

When R-CNN was presented (Girshick et al., 2014), it was one of the first successful proposal-based object detection networks operating at acceptable computational cost. Since then, there have been lots of improvements in object detection, and the principles in R-CNN formed a foundation for proposal-based object detection that still is used in remote sensing analysis. For grid-based object detection, YOLO (Redmon et al., 2016) and SSD (Liu et al., 2016a) formed such foundations recurring in modern grid-based detectors. Traditionally, grid-based object detectors were too elementary to be used on large scale remote sensing images.

2. This paragraph is made a subsection for consistency with architectures that will be introduced in the following sections.

Liu et al. (2017a) presented the DRBox estimating rotatable bounding boxes³, with an architecture based on SSD. Recall section 5.2.5 and Figure 5.4 stressing how great an improvement there is when using RBox over the traditional BBox. Since then, several descendants have been practicing RBoxes.

In chapter 1, it was mentioned that Liu et al. (2018) claimed that there is less focus on detecting orientation of objects in remote sensing images because the great benchmark data sets have omitted data on object orientation, and hence operates with traditional BBoxes. In 2018, Kaggle published a data set on ship detection in optical satellite images (Kaggle, 2018), and Xia et al. (2018) published a data set for object detection in aerial images, opening up for a wider interest in the research field.

Object detection in large scale remote sensing scenes is different from natural everyday scenes, and different methods are often used for these scenarios. The large difference between these scenarios is the large scale and the great variety in object rotation and scale in remote sensing images⁴.

6.3.1 Proposal-based Methods

From section 5.2, it is clear that proposal-based object detection developed rapidly in the initial phase after R-CNN (Girshick et al., 2014) was presented. Most proposal-based models today follows the foundation of Faster R-CNN (Ren et al., 2015) (introduced in section 5.2), but with a twist evolving it to be specialized for a specific task.

Faster R-CNN (Ren et al., 2015) has been considered as the state-of-the-art object detection architecture also within remote sensing images. Recently presented architectures are specialized on large scale images and to recognize ship characteristics, and have shown state-of-the-art performance on ship detection.

6.3.1.1 HDNN: Chen et al. (2014)

Hybrid Deep Neural Network (HDNN) is specifically designed for detecting vehicles in high-resolution satellite images using traditional BBoxes. The basic idea behind this model is that the data is run through multiple subnetworks at the end of the model, where the different subnetworks operate with different receptive fields. This makes the model able to extract variable-scale features, resulting in a more robust shape prediction when object size varies. It uses sliding window for anchor generation, and is built on an undocumented deep CNN.

3. Liu et al. (2016b) introduced the principle of rotatable bounding boxes in 2016, based on Faster-RCNN (Ren et al., 2015). However, the DRBox from 2017 is a more stable and strong tool.

4. In section 7.2, there will be discussed that ship pixels usually are highly underrepresented in satellite images.

6.3.1.2 Faster R-CNN: Ren et al. (2015)

A conceptual description of Faster R-CNN was given in section 5.2.2. The architecture, training methodology and optimization objective will now be described in detail.

Faster region-based convolutional neural network (Faster R-CNN) has emerged from the ancestors R-CNN and Fast R-CNN, and is an end-to-end trainable architecture consisting of a region proposal network, a regression network, and a classification network. Essentially, the RPN suggests locations and shapes (anchors), the regression branch refines these anchors for a better fit, and the classification branch takes in these arbitrarily shaped regions and evaluates the confidence scores. There are in general several times as many proposed regions than objects in the image. This is resolved by applying a user-defined threshold where anchors having a confidence score lower than this threshold are rejected. Also, NMS is applied to the final result to reduce redundancy.

During training, the RPN will get feedback on which regions that coincide with the ground truth object bounding boxes. Gradually, the RPN will learn which proposed regions were accepted as correct predictions and which were false alarms. The main idea behind Faster R-CNN is to integrate an RPN into the CNN used in Fast R-CNN, and use shared weights between the RPN and the object detection network to decrease the inference time and for faster convergence. Faster R-CNN (Ren et al., 2015) operates with traditional BBoxes. However, there have been developed architectures operating with RBoxes based on the Faster R-CNN architecture (Xia et al., 2018).

Network Structure

Figure 6.1 summarizes the main components and data flow in the Faster R-CNN model. A simple fully convolutional network produces convolutional feature maps. An RPN is applied to this feature map to produce prior BBoxes. In the subfigure on the left hand side, the output of the base CNN is passed both to the RPN and to the later RoI pooling layer. This is referred to as using shared weights, and is argued to decrease the computational cost of region proposal (Ren et al., 2015). VGG (Simonyan and Zisserman, 2014) is used as the base CNN, encouraging the use of a pre-trained network.

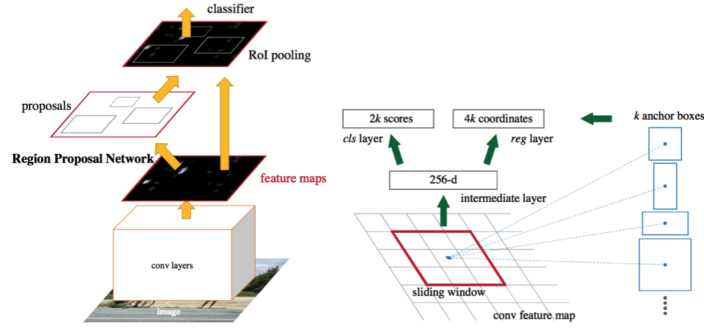


Figure 6.1: Overview of main components in the Faster R-CNN model (left), and a conceptual sketch of the region proposal network (right). Image credit: Ren et al. (2015).

RPN

The right hand side subfigure in 6.1 displays a conceptual sketch of the RPN. A small window is slid over the convolutional feature map, where each sliding window is mapped to a 512-dimensional vector⁵. ReLU activation functions are applied to the small sliding window outputs. This 512-dimensional vector in each sliding window is fed into two fully-connected sibling layers: a BBox-regression layer (*reg*) and a BBox-classification layer (*cls*). Each sliding window predicts k anchors, where k is a hyperparameter, leading to $2k$ and $4k$ parameters output from the *cls* and the *reg* layers, respectively. Experimentation with 3 scales and 3 aspects ratios results in $k = 9$. The effective receptive field when using VGG as the base CNN is 228 pixels (Ren et al., 2015).

Training

$$L_{Faster} = \frac{1}{N_{cls}} \sum_{i=1}^{N_{box}} L_{cls}(\hat{c}^{(i)}, c^{(i)}) + \frac{\lambda}{N_{box}} \sum_{i=1}^{N_{box}} c^{(i)} L_1^{smooth}(\mathbf{l}^{(i)} - \mathbf{g}^{(i)}) \quad (6.1)$$

Equation 6.1 expresses the overall Faster R-CNN loss. N_{cls} and N_{box} are batch size and number of anchors, respectively, $\hat{c}^{(i)}$ is the anchor confidence score, $c^{(i)}$ is the binary ground truth describing if the anchor is an object. \mathbf{l} is predicted BBox parameterization and \mathbf{g} is the ground truth BBox parameterization of the box associated with \mathbf{l} . L_{cls} penalizes false alarms accepted as true objects, and vice versa. L_{cls} is the cross entropy loss, as was introduced back in equation

5. The "256-d" label in 6.1 corresponds to a lighter network than VGG.

4.11, and is now simplified in equation 6.2.

$$L_{cls}(\hat{c}^{(i)}, c^{(i)}) = -c^{(i)} \log(\hat{c}^{(i)}) - (1 - c^{(i)}) \log(1 - \hat{c}^{(i)}) \quad (6.2)$$

L_1^{smooth} is the smooth L_1 loss, claimed to be less sensitive to outliers, expressed in equation 6.3.

$$L_1^{smooth}(\psi) = \begin{cases} 0.5\psi^2 & , |\psi| < 1 \\ |\psi| - 0.5 & , otherwise \end{cases} \quad (6.3)$$

The relation between the BBox parameterization $\mathbf{l} = \{l_x, l_y, l_w, l_h\}$ and our known BBox representation $\{\theta_x, \theta_y, \theta_w, \theta_h\}$, is by the mapping given in equation 6.4.

$$\begin{aligned} l_x &= (\theta_x - x_a)/w_a, & l_y &= (\theta_y - y_a)/h_a \\ l_w &= \log(\theta_w/w_a), & l_h &= \log(\theta_h/h_a) \end{aligned} \quad (6.4)$$

In equation 6.4, $\theta_x, \theta_y, \theta_w$ and θ_h denote the box center (coordinates θ_x and θ_y), width (θ_w) and height (θ_h). θ_x is the predicted x-coordinate and x_a is the anchor x-coordinate. The same applies for $\theta_y, \theta_w, \theta_h$ and the ground truth parameterization \mathbf{g} .

6.3.1.3 R²PN: Zhang et al. (2018)

The rotated region proposal network R²PN is end-to-end trainable and concerns detection of arbitrarily oriented ships. Zhang et al. (2018) introduced both a rotated RPN, and a rotated RoI pooling layer. The rotated RPN uses the acknowledged VGG network (Simonyan and Zisserman, 2014) for feature extraction, followed by numerous user defined sizes, aspect ratios and angles combined with the feature map to produce the region proposals. These are again regressed to produce more accurate final predictions. The rotated RoI pooling layer acts as a rotated max pooling operator, and reduces to a standard RoI pooling if there is no rotation. This model is really interesting and addresses a lot of the concepts tackled in this thesis, but is not included in the experiments due to time limitations as since no source code is publically available.

6.3.1.4 OBB for Faster R-CNN: Xia et al. (2018)

Faster R-CNN (Ren et al., 2015) has also be extended to predict oriented bounding boxes (OBB)⁶. This model differs from other models predicting arbitrary-oriented objects, as it predicts all corners using eight parameters,

6. OBB is another name for RBox.

instead of the common $\theta_{RBox} = \{\theta_x, \theta_y, \theta_w, \theta_l, \theta_\phi\}$.

Xia et al. (2018) do also use a version of rotated RoI pooling, but is not sharing the model details. The use of eight parameters over the more common five parameter approach for describing a RBox has later been studied further. Experiments have shown that eight parameters result in less constraints for object shape and can potentially enclose objects better. However, the lack of constraint often result in chaotic predictions for small and challenging objects, and the field of research has mostly reverted back to the five parameter standard (Yang et al., 2018).

6.3.1.5 R-DFPN: Yang et al. (2018)

The Rotation Dense Feature Pyramid Network (R-DFPN) aims to address the problem of ships being hard to detect if the ship width is narrow. It is built on the principles of the original Feature Pyramid Network (FPN) (Lin et al., 2017b). Ships tend to appear in various shapes and sizes in remote sensing images. This model is therefore trained using a multi-scale detection framework. This will encourage the model to reuse training features at different scales. Yang et al. (2018) also introduces a multi-scale RoI alignment layer, to compensate for the RoI pooling layer present in proposal-based object detection models. Standard RoI is argued to have problems handling large aspect ratios and to cause misalignment. RoI alignment is much like RoI pooling, but is done without quantifying the feature grid. This is claimed to result in narrow objects being more highlighted.

6.3.1.6 SRDet: Yang et al. (2019)

Yang et al. (2019) presented an object detector specialized for detecting small, cluttered and rotated objects (SCRDet)⁷. This model is end-to-end trainable, and the three subnetworks, SF-Net, MDA-Net and Rotation Branch, constitute the overall SCRDet as a pipeline. This architecture is more complex than the ones studied so far, and involves i.a. multi-dimensional attention to highlight small out-of-focus objects. Even though the complexity leads to higher precision, the inference and training time will also reflect the complexity.

The subnetwork SF-Net incorporates feature fusion and performs detailed region proposal. MDA-Net is a multi-dimensional attention network specialized for highlighting pixels and channels of interest and to neglect noisy background areas. The Rotation Branch introduces and regresses a rotation parameter for the region proposals, opening up for RBox prediction.

7. It also goes under the name R2CNN++ as it is an extension from the Rotational Region CNN (R2CNN) presented in Jiang et al. (2017).

6.3.2 Grid-based Methods

The evolution of grid-based object detection started really to accelerate, as described earlier, when YOLO (Redmon et al., 2016) and SSD (Liu et al., 2016a) were presented. These models build the foundation of grid-based object detection, and their principles are found in modern grid-based object detectors.

6.3.2.1 DRBox: Liu et al. (2017a)

The Detector using RBox (DRBox) was of the first successful object detector using RBoxes, and is established as a simple and innovative model giving state-of-the-art results for a significant period after it was published. DRBox is extended from the SSD detection framework, as briefly introduced in section 5.2.3, to involve angle estimation.

DRBox is not fully end-to-end trainable, but its performance is pseudo-rotation invariant. To achieve this, the model is trained on a wide variety of differently rotated objects to learn that the object can be detected with any orientation. In addition, by including the angle parameter in the predicted bounding boxes, the model can realize the existence of orientation differences, rather than being confused by rotation (Liu et al., 2017a).

Network Structure

In equation 4.16, the standard IoU criterion for two intersecting BBoxes was introduced. The IoU criterion is used in this architecture for NMS. However, this model first proposes a prior RBox, then regresses it until it coincides with the ground truth RBox. Because the intersection of two RBoxes can be any polygon with no more than eight sides, the traditional IoU criterion is a weak tool (Liu et al., 2017a). The angle-related IoU (ArIoU) is therefore introduced as a criterion for refining the angle parameter in the prior RBox. The ArIoU between a predicted RBox Ψ and a ground truth RBox Γ is defined in equation 6.5.

$$ArIoU(\Psi, \Gamma) = \frac{area(\widehat{\Psi} \cap \Gamma)}{area(\widehat{\Psi} \cup \Gamma)} \cos(\theta_{\phi, \Psi} - \theta_{\phi, \Gamma}) \quad (6.5)$$

In equation 6.5, $\theta_{\phi, \Psi}$ and $\theta_{\phi, \Gamma}$ are the angles of regions Ψ and Γ , respectively. $\widehat{\Psi}$ is identical to Ψ , except having angle parameter $\theta_{\phi, \Gamma}$ instead of $\phi_{\phi, \Psi}$. This architecture is not distinguishing between bow and stern of the ship i.e., if the predicted RBox has an angle $\theta_{\phi, \Psi}$, then the true ship heading is predicted to lie in the set $\{\theta_{\phi, \Psi}, \theta_{\phi, \Psi} \pm 180^\circ\}$. Because of this, the authors also presented the $ArIoU_{180}$ measure (defined in equation 6.6) to better tackle the situations where $\theta_{\phi, \Psi} - \theta_{\phi, \Gamma} \approx 180^\circ$. Note that equations 6.5 and 6.6 is non-commutative

(i.e. $ArIoU(\Psi, \Gamma) \neq ArIoU(\Gamma, \Psi)$). The standard IoU criterion is commutative.

$$ArIoU_{180}(\Psi, \Gamma) = \frac{area(\widehat{\Psi} \cap \Gamma)}{area(\widehat{\Psi} \cup \Gamma)} | \cos(\theta_{\phi, \Gamma} - \theta_{\phi, \Gamma}) | \quad (6.6)$$

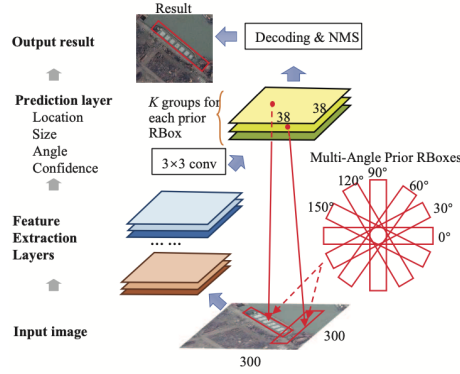


Figure 6.2: Network structure of DRBox (Liu et al., 2017a). The model searches for objects by sliding and rotating prior RBoxes on input images and return location and orientation of objects.

Figure 6.2 presents the network structure of DRBox, with its convolutional architecture. A truncated VGG-net (Simonyan and Zisserman, 2014) is used for feature extraction. The last convolutional layer is for prediction and includes K groups of channels, where K is the number of prior RBoxes at each location. All prior RBox locations are determined from a predefined grid. For each prior RBox, the prediction layer outputs a predicted confidence, and 5 corresponding parameters describing the regressed offset from the prior RBox. At last, there is a decoding process for moving the prior RBoxes to the refined position, and NMS to remove repeated predictions. To moderate the abundance of prior RBoxes, the user predefines which angles, aspect ratios and sizes to be proposed at each location.

The receptive field of DRBox is 108 pixels (Liu et al., 2017a). With the constantly evolving remote sensing technology, the reader may notice that this will put some restrictions.

During training, a prior RBox P is assigned to match the corresponding ground truth RBox G iff $ArIoU(P, G) > 0.5$. These matched boxes will represent positive samples and contribute the loss during training⁸. The objective loss function is an extension from SSD (Liu et al., 2016a), where there is included

8. Note that there still is an abundance of matching RBoxes. Many of these will be removed by NMS.

an additive angle-related term.

$$L(x, \mathbf{c}, \hat{\boldsymbol{\theta}}_{RBox}, \boldsymbol{\theta}_{RBox}) = \frac{1}{N} \left(L_{conf}(\mathbf{c}) + L_{rbox}(x, \hat{\boldsymbol{\theta}}_{RBox}, \boldsymbol{\theta}_{RBox}) \right) \quad (6.7)$$

The overall objective loss function is given in equation 6.7, where N is the number of matched prior RBoxes. $L_{conf}(\mathbf{c})$ is the softmax loss over all positive and negative samples, kept in the confidence vector \mathbf{c} . The RBox regression loss $L_{rbox}(x, \hat{\boldsymbol{\theta}}_{RBox}, \boldsymbol{\theta}_{RBox})$ between the predicted RBox $\hat{\boldsymbol{\theta}}_{RBox}$ and the ground truth RBox $\boldsymbol{\theta}_{RBox}$ is given in equation 6.8.

$$L_{rbox}(x, \hat{\boldsymbol{\theta}}_{RBox}, \boldsymbol{\theta}_{RBox}) = \sum_{i \in Pos} \sum_j \sum_{m \in \{x, y, w, l, \phi\}} x_{ij} L_1^{smooth} \left(\bar{\theta}_{RBox, m}^{(i)} - \bar{\theta}_{RBox, m}^{(j)} \right) \quad (6.8)$$

Equation 6.8 contains membership coefficients $x_{ij} \in \{0, 1\}$ being 1 iff the i -th prior RBox is matched to the j -th ground truth RBox. Further, $\bar{\theta}_{RBox}$ and $\tilde{\theta}_{RBox}$ are the regressed offsets of all parameters in $\hat{\boldsymbol{\theta}}_{RBox}$ and $\boldsymbol{\theta}_{RBox}$, corresponding to the prior RBox p , as defined in equation 6.9. L_1^{smooth} is the smooth L_1 loss, also used in Faster R-CNN, and expressed in equation 6.3. By minimizing the overall objective loss function during training, and hence the angle regression term \hat{t}_ϕ , it ensures that the correct angle is learned during training.

$$\begin{aligned} \hat{t}_x &= (t_x - p_x)/p_w, & \hat{t}_y &= (t_y - p_y)/p_h \\ \hat{t}_w &= \log(t_w/p_w), & \hat{t}_h &= \log(t_h/p_h) \\ \hat{t}_\phi &= \tan(t_\phi - p_\phi) \end{aligned} \quad (6.9)$$

Complementary Details

It was mentioned that the angle, aspect ratio and size needs to be user defined. In this thesis, we are only concerned with ship objects and typical aspect ratios and sizes repeat, but in the general case this is considered a drawback.

6.3.2.2 YOLOv2: Redmon and Farhadi (2017)

In chapter 5, YOLOv1⁹ was conceptually introduced as a benchmark grid-based object detector. YOLO version two (YOLOv2) is an extension of YOLOv1, with the objective of significantly improving accuracy in addition to reducing inference time. Main design components included for increased performance

9. As extensions of YOLO have been developed, the traditional YOLO is often referred to as YOLO version one (YOLOv1).

includes batch normalization, location and scale offset regression, and a fully convolutional structure.

Where other object detectors tend to be large, complex and precise, YOLOv2 simplifies the network even more from YOLOv1, and includes a collection of carefully selected promising principles, like the mentioned ones, to achieve fast and accurate predictions. YOLOv2 predicts traditional BBoxes. It is not rotation invariant, however, rotation invariance is commonly synthesized using the approach described in section 5.2.4.

Network Structure

Similarly as for YOLOv1, the image is split into a grid with C^2 cells. The cells can be thought of as prior BBoxes, where both shape and localization is learnable regression parameters to enclose objects better.

Table 6.1 presents all layers of the YOLOv2 object detector, with shapes corresponding to experiments performed in part IV. There are seven blocks of convolutional layers, with ReLU activation function after each convolutional layer, and MaxPooling between each block. All shapes are stated according to experiments performed in part IV, and the output layer corresponds to predicting 1024 BBoxes described by 5 parameters $\theta_{BBox} = \{\theta_x, \theta_y, \theta_w, \theta_h, c\}$. The output shape of the last layer can be modified to fit various problems. It is also common to apply an activation function to the final output, dependent on the task under consideration.

The final BBoxes predicted by YOLOv2 is calculated from equation 6.10 (Redmon and Farhadi, 2017). $\theta_{BBox} = \{\theta_x, \theta_y, \theta_w, \theta_h\}$ is the predicted bounding box, t_x, t_y, t_w and t_h are predicted offset values, c is the confidence score, t_o is outputted confidence before sigmoid activation function is applied¹⁰, p_x and p_y is the top left corner of the cell in the grid, and p_w and p_h is the height and width of the cell, respectively. All $\theta_x, \theta_y, \theta_w, \theta_h, p_x, p_y, p_w$ and p_h are normalized relative to the image width and height. By using this approach, the localization will be predicted relative to each cell and the ground truths will always have a position relative to each cell bounded on the range $[0, 1]$. The sigmoid activation function is applied to t_x and t_y in equation 6.10 to constrain the predicted locations relative to the cells as well. Using a per-cell location approach like this remedies model instability and results in more sensible predicted offsets at an early stage of training.

10. Applying sigmoid to achieve a confidence score $c \in [0, 1]$ is very common. This allows the confidence scores to be interpreted as probabilities.

Table 6.1: All layers with corresponding properties of the YOLOv2 network (Redmon and Farhadi, 2017). Output sizes correspond to image shapes used for experimentation in this thesis.

Type	Filters	Filter Size / Stride	Output Size
Conv.	32	3×3	512×512
MaxPool		$2 \times 2 / 2$	256×256
Conv.	64	3×3	256×256
MaxPool		$2 \times 2 / 2$	128×128
Conv.	128	3×3	128×128
Conv.	64	1×1	128×128
Conv.	128	3×3	128×128
MaxPool		$2 \times 2 / 2$	64×64
Conv.	256	3×3	64×64
Conv.	128	1×1	64×64
Conv.	256	3×3	64×64
MaxPool		$2 \times 2 / 2$	32×32
Conv.	512	3×3	32×32
Conv.	256	1×1	32×32
Conv.	512	3×3	32×32
Conv.	256	1×1	32×32
Conv.	512	3×3	32×32
MaxPool		$2 \times 2 / 2$	16×16
Conv.	1024	3×3	16×16
Conv.	512	1×1	16×16
Conv.	1024	3×3	16×16
Conv.	512	1×1	16×16
Conv.	1024	3×3	16×16
Conv.	1024	1×1	5×1024

$$\begin{aligned}
 \theta_x &= \sigma(t_x) + p_x \\
 \theta_y &= \sigma(t_y) + p_y \\
 \theta_w &= p_w e^{t_w} \\
 \theta_h &= p_h e^{t_h} \\
 c &= \sigma(t_o)
 \end{aligned}
 \tag{6.10}$$

$$\begin{aligned}
L_{YOLOv2} &= L_{localization} + L_{confidence} \\
&= \lambda_{coord} \sum_{i=0}^{C^2} \sum_{j=0}^B \mathbb{1}_{ij}^{obj} \left[\left(\theta_x^{(i)} - \hat{\theta}_x^{(i)} \right)^2 + \left(\theta_y^{(i)} - \hat{\theta}_y^{(i)} \right)^2 \right] \\
&+ \lambda_{coord} \sum_{i=0}^{C^2} \sum_{j=0}^B \mathbb{1}_{ij}^{obj} \left[\left(\sqrt{\theta_w^{(i)}} - \sqrt{\hat{\theta}_w^{(i)}} \right)^2 + \left(\sqrt{\theta_h^{(i)}} - \sqrt{\hat{\theta}_h^{(i)}} \right)^2 \right] \\
&+ \sum_{i=0}^{C^2} \sum_{j=0}^B \mathbb{1}_{ij}^{obj} \left(c^{(i)} - \hat{c}^{(i)} \right)^2 \\
&+ \lambda_{noobj} \sum_{i=0}^{C^2} \sum_{j=0}^B \mathbb{1}_{ij}^{noobj} \left(c^{(i)} - \hat{c}^{(i)} \right)^2
\end{aligned} \tag{6.11}$$

The detection loss for YOLOv2 origins from Sum of Squared Errors¹¹ and is given in equation 6.11¹², where the following notation is used:

- λ_{coord} and λ_{noobj} are hyperparameters to regulate the weighting of coordinate loss in the total loss, and to decrease the total loss when detecting only background, respectively.
- C^2 and B are the total number of cells in the grid, and total number of bounding boxes, respectively.
- $\mathbb{1}_{ij}^{obj}$ is a boolean value indicating if the j -th bounding box in cell i is responsible for detecting the object. $\mathbb{1}_{ij}^{noobj}$ is the complement of $\mathbb{1}_{ij}^{obj}$.
- $\hat{c}^{(i)}$ is the confidence score of box j in cell i .
- $\{\theta_x^{(i)}, \theta_y^{(i)}, \theta_w^{(i)}, \theta_h^{(i)}, c^{(i)}\}$ describes the parameters of the true bounding box j in cell i .
- $\{\hat{\theta}_x^{(i)}, \hat{\theta}_y^{(i)}, \hat{\theta}_w^{(i)}, \hat{\theta}_h^{(i)}, \hat{c}^{(i)}\}$ describes the parameters of the predicted bounding box j in cell i , calculated from equation 6.10.

11. Similar to MSE, introduced in section 4.4.

12. The loss can be extended to include classification loss as well. This is omitted as it is not relevant for this thesis.

YOLO will predict multiple BBoxes per cell¹³. Only the best prediction is used when calculating the loss. The best prediction is the BBox having largest IoU with a ground truth label, and is referred to as the *responsible* prediction for that object. This strategy will cause the model to make better and more accurate predictions for every iteration. The localization loss measures the error between the ground truth object and the responsible prediction. Square root of the width and height is used to address the problem of weighing the absolute error of small and large boxes unequally. The confidence loss is split into one part handling predictions in cells containing ground truth objects, and one part handling cells where there are no ground truth objects. The binary variable $\mathbb{1}_{ij}^{obj}$ and its complement $\mathbb{1}_{ij}^{noobj}$ keeps track of which predictions that are considered responsible. The hyperparameter λ_{noobj} remedies the class imbalance problem arising from training data having a large majority of empty cells.

The powerful regularization technique of batch normalization is applied on all convolutional layers (Redmon and Farhadi, 2017). NMS is performed at the end of the model before outputting the final results during inference and validation. Royer and Lampert (2020) suggests that $\lambda_{coord} = 1$ and $\lambda_{noobj} = 5$ are appropriate values for the common remote sensing data set.

6.3.2.3 Tiny YOLO

Originating from YOLOv2, the shallower model Tiny YOLO (or Tiny YOLOv2) has also been developed. No official article has been posted on Tiny YOLO, but except having a lighter neural network as base, all concepts is identical as for YOLOv2¹⁴.

Tiny YOLO is specialized for detecting simpler objects at high speed. The lightweight neural network being the foundation of Tiny YOLO is presented in Table 6.2. It has seven convolutional layers with subsequent ReLU activation functions and MaxPool layers in between. Similar as for YOLOv2 in Table 6.1, this model also is a fully convolutional network, where the output shape of the last layer is customized for the specific problem.

Tiny YOLO reports exceptional frame rate during inference. It is able to run over three times faster than YOLOv2, but in return the mAP decreases with about 25%¹⁵.

13. https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088 summarizes different YOLO models with their corresponding loss functions perfectly.

14. Tiny YOLO is developed by the same developer as YOLOv2. The official web cite, with related experiments, can be found at <https://pjreddie.com/darknet/yolo/>.

15. Tiny YOLO operates at unbeatable 207 fps whereas YOLOv2 operates at 67 fps, according to the official YOLO page: <https://pjreddie.com/darknet/yolov2/>.

Table 6.2: All layers with corresponding properties of the tiny-YOLO network. Output sizes correspond to image shapes used for experimentation in this thesis.

Type	Filters	Filter Size / Stride	Output Size
Conv.	16	3×3	51×512
MaxPool		$2 \times 2 / 2$	25×256
Conv.	32	3×3	25×256
MaxPool		$2 \times 2 / 2$	12×128
Conv.	64	3×3	12×128
MaxPool		$2 \times 2 / 2$	64×64
Conv.	128	3×3	64×64
MaxPool		$2 \times 2 / 2$	32×32
Conv.	256	3×3	32×32
MaxPool		$2 \times 2 / 1$	16×16
Conv.	512	3×3	16×16
Conv.	1024	1×1	5×1024

6.3.3 ODGI: Royer and Lampert (2020)

In December 2019, Prof. Christoph Lampert presented how his ph.d. student Amélie Royer and he utilizes object groupings in their Object Detection with Grouped Instances (ODGI) model at a deep learning conference in Tromsø. This brought the author to the idea of testing this model on ship detection in optical satellite images as part of this thesis. Royer and Lampert (2020) has primarily focused on detecting people and cars in aerial images, not ships in satellite images. ODGI will be experimented on in chapter 9, and is the foundation behind the novel model presented in chapter 8.

The main principle of ODGI is to combine a series of stages where each stage is an individual object detector, able to predict either individual object instances or grouped object instances to pass further to consecutive stages. Royer and Lampert (2020) experiments on using the grid-based YOLOv2 (Redmon and Farhadi, 2017), Tiny YOLO and MobileNet (Sandler et al., 2018) as object detectors in each stage. Because each stage is capable of predicting either an individual instance or to propose a grouped instance for further analysis in consecutive stages, ODGI can be considered a hybrid of grid-based and proposal-based object detector, where each stage that can propose grouped instances is a region proposal network. When focusing on individual or grouped instances at an early stage, most of the computation is concentrated on important and decisive regions (Royer and Lampert, 2020).

Most existing object detectors today are inefficient for grouped instances and do not exploit the potentials in grouped instances (Royer and Lampert, 2020). The motivation behind this pipeline scheme is that groupings are visually

more salient and easier to detect than individuals. ODGI is a single-class object detection scheme for inhomogeneously distributed objects, but can easily be extended to include multi-object categories and classification as well.

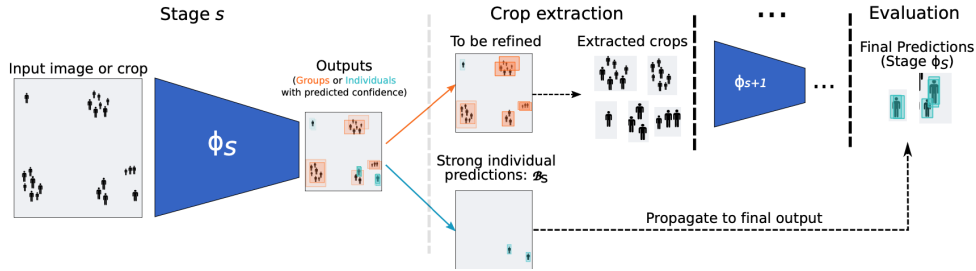


Figure 6.3: ODGI overview. Compiled by a grid-based object detector at each stage ϕ_s . A stage ϕ_s can predict an individual object instance or propose a grouped object instance to be passed to consecutive stages. Image credits: Royer and Lampert (2020)

The multi-stage pipeline visualized in Figure 6.3 consists of S stages, $\phi_S \circ \dots \circ \phi_1$, $S > 1$. Each stage ϕ_s for $s < S - 1$, is a lightweight object detection network predicting both individual object instances and grouped object instances (Royer and Lampert, 2020). A candidate for further refinement predicted at stage ϕ_s will be the input to the consecutive stage, ϕ_{s+1} . Such a segment, input to the consecutive stage, will be denoted as a *crop*. The last stage is constrained to only output individual objects.

For sorting the predictions at each stage, there is a need for two hyperparameters, τ_{low} and τ_{high} , and a binary group flag g , which will be introduced in the next paragraph. There are three possibilities during this sorting:

- i) $c \leq \tau_{low}$: The prediction is discarded.
- ii) $c \geq \tau_{high}$ and $g = 0$: The prediction is considered as an individual object instance and skips consecutive stages.
- iii) ($c > \tau_{low}$ and $g = 1$) or ($\tau_{low} \leq c < \tau_{high}$ and $g = 0$): Prediction is either a group or an individual with medium confidence and is passed on to the consecutive stage.

NMS is applied to all candidates of possibility iii) at intermediate stages before being passed on to next stage. The hyperparameter τ_s regulates the maximum number of predictions to refine in consecutive stages. After NMS, these τ_s predictions will have highest possible confidence and lowest possible overlap. These predictions are processed to form subimages ready for further refinement in stage ϕ_{s+1} . This process includes multiplying the prediction width and height by $1/o_w$ and $1/o_h$, respectively, where (o_w, o_h) are learned offset values as describes below. The final output will be all accepted predictions of the final stage ϕ_S , in addition to all strong individual predictions (possibility ii))

in preceding stages.

Training

The model is trained to recognize individuals and groups in the respective stages. The original labels are therefore expanded to form group labels as well, which is used as ground truth when training the model. To expand the ground truth labels from the individual object instance form to the grouped object instance form, there is a need for a study of adjacent individuals. This is done by utilizing the grid that was formed mainly to be used in the grid-based object detectors over the stages. This grid size is dependent on image resolution and the average object sizes. Royer and Lampert (2020) uses a grid size 32 times larger than the pixel size. If two individual objects appear in the same grid cell, a group mask is formed to enclose both these individuals, and possibly additional adjacent individuals. An example of one such group mask is visualized in Figure 6.4. The individual object labels for both these ships are occurring in at least one common cell, and is therefore considered as a group instance. Individual object instances with no adjacent objects will retain its original label.

In addition to the common practice of using five parameters for each prediction ($\theta_{BBox} = \{\theta_x, \theta_y, \theta_w, \theta_h, c\}$), ODGI is augmented to also predict a binary group flag g , as well as two offset values (o_w, o_h). g indicates whether the prediction is considered to be an individual instance ($g = 0$) or a group instance ($g = 1$). The offset values (o_w, o_h) are used for rescaling group instances before passing these to subsequent stages. These offset values will ensure that relevant regions are covered well enough, and compensate for the potential challenge that the detectors are learned to exactly predict the ground truth labels, not to fully enclose them. Intermediate stages are also modified to predict one BBox per cell. The intuition is that the grid resolution is not always fine enough to contain only one object per cell. If that was fulfilled, the problem is reduced to a simple object detection problem. Otherwise, if a cell contains multiple objects, the cell needs a closer study to distinguish individual object instances.

Each ODGI stage is trained independently using the three loss terms expressed in equation 6.12, and is optimized using standard backpropagation.

$$L_{ODGI} = L_{groups} + L_{coords} + L_{offsets} \quad (6.12)$$

The loss terms will be described in the following.

The group loss term, L_{groups} , will drive the model to classify predictions as individuals or groups, and is calculated as a binary classification objective

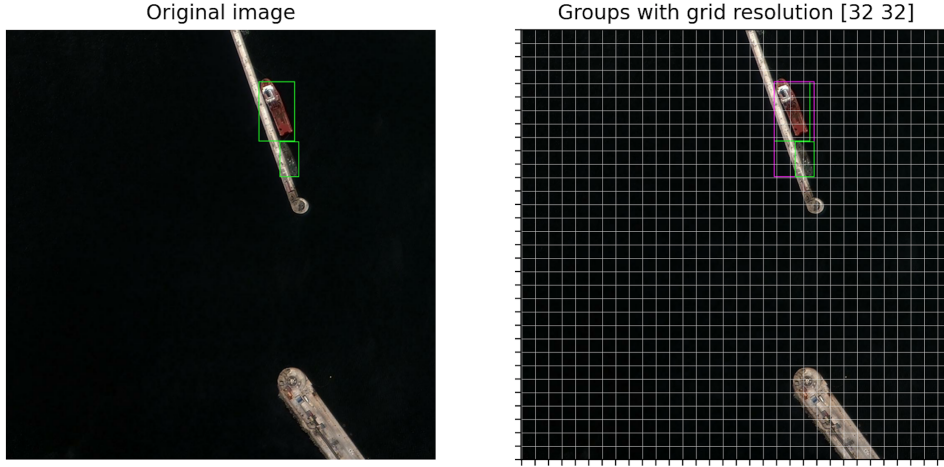


Figure 6.4: Visualization of how a group mask (purple) is formed based on two single instance masks (green). The single instance masks are both appearing in at least one common cell and are therefore considered as members of a group instance. The grid (right hand side) consists of cells of 32×32 pixels and is used for grid-based detection.

expressed by equation 6.13:

$$L_{groups} = - \sum_{i,j} A^{(i,j)} \left(\bar{g}^{(j)} \log(g^{(i,j)}) + (i - \bar{g}^{(i,j)}) \log(1 - g^{(i,j)}) \right) \quad (6.13)$$

$A^{(i,j)}(n)$ is a binary mask indicating if an individual ground truth label $b^{(n)}$, $n = 1, \dots, N$ is assigned to output cell (i, j) . Mathematically expressed by equation 6.14:

$$A^{(i,j)}(n) = \llbracket |b^{(n)} \cap cell^{(i,j)}| > 0 \rrbracket, \quad \text{with} \llbracket x \rrbracket = 1 \text{ if } x, \text{ else } 0 \quad (6.14)$$

Formally, $B^{(i,j)}$ is now introduced as the predictor to cell (i, j) , and will be used in the following. We also denote its ground truth coordinates as $\bar{B}^{(i,j)}$ and its group flag as $\bar{g}^{(i,j)}$, as is used in equation 6.13, mathematically expressed as:

$$\bar{B}^{(i,j)} = \bigcup_{n | A^{(i,j)}(n)=1} b^{(n)} \quad (6.15)$$

$$\bar{g}^{(i,j)} = \llbracket \#\{n | A^{(i,j)}(n) = 1\} > 1 \rrbracket \quad (6.16)$$

In equation 6.15, \bigcup denotes the minimum enclosing bounding box of the set.

The coordinate loss term of equation 6.12, L_{coords} , is based on a standard mean squared error loss, similar to what is used in for example YOLOv2. It is mathematically expressed in equation 6.17 using the terms and sets introduced

for this model:

$$L_{coords} = \sum_{i,j} A^{(i,j)} \|B^{(i,j)} - \bar{B}^{(i,j)}\|^2 + \lambda_{conf} \|c^{(i,j)} - \bar{c}^{(i,j)}\|^2 + \lambda_{noobj} \sum_{i,j} (1 - A^{(i,j)})(c^{(i,j)})^2 \quad (6.17)$$

The two first terms of equation 6.17 are least square error objectives of predicted coordinates and confidence scores. The ground truth confidence score \bar{c}^{ij} is defined as the IoU between the prediction and its assigned ground truth target, expressed by equation 6.18:

$$\bar{c}^{(i,j)} = IoU(B^{(i,j)}, \bar{B}^{(i,j)}) = \frac{|B^{(i,j)} \cap \bar{B}^{(i,j)}|}{|B^{(i,j)} \cup \bar{B}^{(i,j)}|} \quad (6.18)$$

The last term in equation 6.17 is for penalizing non-zero confidence scores in empty cells. This is necessary for remedying the class imbalance problem, similar to what was used in the YOLOv2 loss in equation 6.11. The parameters λ_{conf} and λ_{noobj} are penalizing hyperparameters to be adjusted for the specific problem.

The offset loss term in equation 6.12, $L_{offsets}$, will encourage refinement of prior boxes at intermediate stages, and hence better coverage of extracted crops. Intermediate stages will predict offset values (o_w, o_h) that are used for rescaling crops before they are passed on to consecutive stages, as described earlier.

$$L_{offsets} = \sum_{i,j} A^{(i,j)} \left[\left(o_w - \bar{o}_w(B^{(i,j)}, \bar{B}^{(i,j)}) \right)^2 + \left(o_h - \bar{o}_h(B^{(i,j)}, \bar{B}^{(i,j)}) \right)^2 \right] \quad (6.19)$$

Equation 6.19 expresses how the offset loss term is calculated, where the target vertical and horizontal offset values, $\bar{o}_w(B^{(i,j)}, \bar{B}^{(i,j)})$ and $\bar{o}_h(B^{(i,j)}, \bar{B}^{(i,j)})$, respectively, are determined as:

$$\bar{o}_h(B^{(i,j)}, \bar{B}^{(i,j)}) = \max \left(1, h(B^{(i,j)}) / h^{scaled}(B^{(i,j)}, \bar{B}^{(i,j)}) \right) \quad (6.20)$$

In equation 6.20, α denotes the center y-coordinate, $h(B^{(i,j)})$ denotes the height of box $B^{(i,j)}$ and δ is a margin hyperparameter regularizing the minimum desirable overlap between a prediction and the corresponding ground truth. Royer and Lampert (2020) uses $\delta = 0.0025$, half the average object size in their data. Further, $B^{(i,j)}$ and $\bar{B}^{(i,j)}$ denotes a predicted box and its assigned ground

truth box, respectively. The function $h^{scaled}(B^{(i,j)}, \bar{B}^{(i,j)})$ scales the prediction $B^{(i,j)}$ with respect to its assigned ground truth $\bar{B}^{(i,j)}$, and is calculated as:

$$h^{scaled}(B^{(i,j)}, \bar{B}^{(i,j)}) = \max \left(\left| (\alpha(\bar{B}^{(i,j)}) + h(\bar{B}^{(i,j)})/2 + \delta) - \alpha(B^{(i,j)}) \right|, \right. \\ \left. \left| (\alpha(\bar{B}^{(i,j)}) - h(\bar{B}^{(i,j)})/2 - \delta) - \alpha(B^{(i,j)}) \right| \right) \quad (6.21)$$

The horizontal offset \bar{o}_w is calculated identically as \bar{o}_h , but using the center x-coordinate and its width instead.

Complementary Details

The number of stages S to be used in the pipeline is user defined and should be set according to image resolution and object sizes, with the speed versus accuracy trade-off in mind. Royer and Lampert (2020) argues that $S = 2$ is sufficient for most ordinary remote sensing problems, and that $S > 2$ is more suited for very-large scale scenarios, e.g. gigapixel images.

In total, the ODGI model is modified using the hyperparameters

$\tau_{low}, \tau_{high}, \tau_{nms}, \tau_s, S, \lambda_{conf}, \lambda_{noobj}$ and δ . Recall that τ_{low} and τ_{high} are the weak (τ_{low}) and strong (τ_{high}) confidence thresholds, τ_{nms} is the NMS threshold, τ_s is the number of crops to evaluate at stage ϕ_s and S is the total number of stages in the pipeline. The hyperparameters λ_{conf} and λ_{noobj} are for weighting loss terms, and δ is a margin hyperparameter regularizing the desirable overlap between a prediction and the corresponding BBox. For the loss penalty terms, Royer and Lampert (2020) successfully uses $\lambda_{conf} = 5, \lambda_{noobj} = 1$ and $\delta = 0.0025$. $S = 2$ stages is used, as their data set is considered as an ordinary remote sensing data set. When using $S = 2$ stages, τ_s is reduced to τ_1 . To force the model to be trained on as many groups and situations as possible, the remaining hyperparameters is split into $\tau_{low}^{train}, \tau_{high}^{train}, \tau_{nms}^{train}, \tau_1^{train}$ during training, and $\tau_{low}^{test}, \tau_{high}^{test}, \tau_{nms}^{test}, \tau_1^{test}$ during evaluation and inference.

Royer and Lampert (2020) states that the motivation behind the model is to decrease grid resolution, and hence the number of anchors, while performance is maintained. Decreasing the grid resolution will offer substantial computational savings, and provides opportunities for running inference on devices having limited computational or energy resources.

6.3.4 Summary

This chapter has reviewed key models in the object detection literature, with a particular focus on ship detection. Traditional machine learning-based models were introduced initially, before main relevant deep learning architectures were described in detail. Faster R-CNN, DRBox, YOLOv2, Tiny YOLO and ODGI are especially noteworthy models, as these will be studied and experimented on further in the thesis.

Part III

Methodology



SuperView Data Set

The data set used for experimentation in this thesis consists of optical satellite images captured by the SuperView satellites. This chapter will describe how the data set is collected and modified, and essential properties will be described.

Five SuperView satellite products captured in near-shore areas around Gibraltar and Perth, Australia are used in the data set. Table 3.1, defined back in section 3.3.2, presented the five bands recorded by the SuperView instrument and their spatial resolution. All data used for experimentation in this thesis are RGB images originating from pansharpened satellite products (Sai et al., 2019). The panchromatic band is used to sharpen the red, green, and blue components to synthesize a spatial resolution of 0.5 m. Only these three RGB components are used in the data set.

The large SuperView satellite products are tiled up (ref. section 3.4) to achieve an appropriate data shape for experimentation. For the experiments in part IV, this corresponds to 1024×1024 images. This image shape is inspired by official experiments on ODGI (Royer and Lampert, 2020). Because of the spherical earth shape, the satellite products will have a non-rectangular shape and some areas will be zero-padded as they are not included in the sensed area. Only tiles with at least 30% non-zero pixels were included in this data set. This will exclude unnecessary empty tiles, whilst still including most ships close to the satellite product boundaries.

Key details of the SuperView data set are summarized as:

- Five satellite products over Perth and Gibraltar, Australia.
- Pansharpned to 0.5 m spatial resolution.
- Total of 3308 annotated ships.
- 380 positive image tiles¹.
- 80 % of the positive tiles are used in the training data set. The remaining 20 % of the positive tiles are divided into one validation data set and one test data set.
- 8.70 is the average number of ships in a tile.
- 683 ships is the maximum number of ships present in one tile.

Ship annotations originates from manually performed near real-time services by KSAT. These annotations consists of position, shape and heading. As a result of corrupted image data during storage, many annotations have been duplicated and displaced subsequently. A significant effort to adapt the original data to a suitable detection training data set has been made as a contribution by this thesis. This adaption includes i.a., annotation refinement, data washing and data set completion. Ships are annotated according to the $\theta_{RBox} = \{\theta_x, \theta_y, \theta_w, \theta_l, \theta_\phi\}$ format, where θ_ϕ points in the ship bow direction relative to image x-axis. Several models used for experimentation in part IV practice traditional BBoxes. The annotation format is transformed as a pre-processing step for these models. A code snippet for doing this mapping can be found in appendix section 14.1. All label parameters are normalized by dividing location and shape parameters by image size and the rotation parameter by 180. This means that all rotation values initially are confined to $[0, 180)$, and that bow and stern of the ship is not distinguished.

A comparison of this SuperView data set relative to other benchmark object detection data sets² shows that 380 positive image tiles are relatively few samples for object detection data sets.

1. A positive image tile is an image tile where at least one annotated ship is present.
2. Other benchmark object detection data sets are for instance Pascal VOC (Everingham et al., 2010), ImageNet (Russakovsky et al., 2015), MS COCO (Lin et al., 2014) and Airbus ship detection (Kaggle, 2018).

7.1 SuperView Appearances

Environmental conditions cause artifacts in optical satellite images, and generate a diversity in the SuperView image appearance. Such environmental effects are typically sea waves, time of day and weather. All satellite products used in this data set are free of clouds, enabling the potential for all ships being visible.

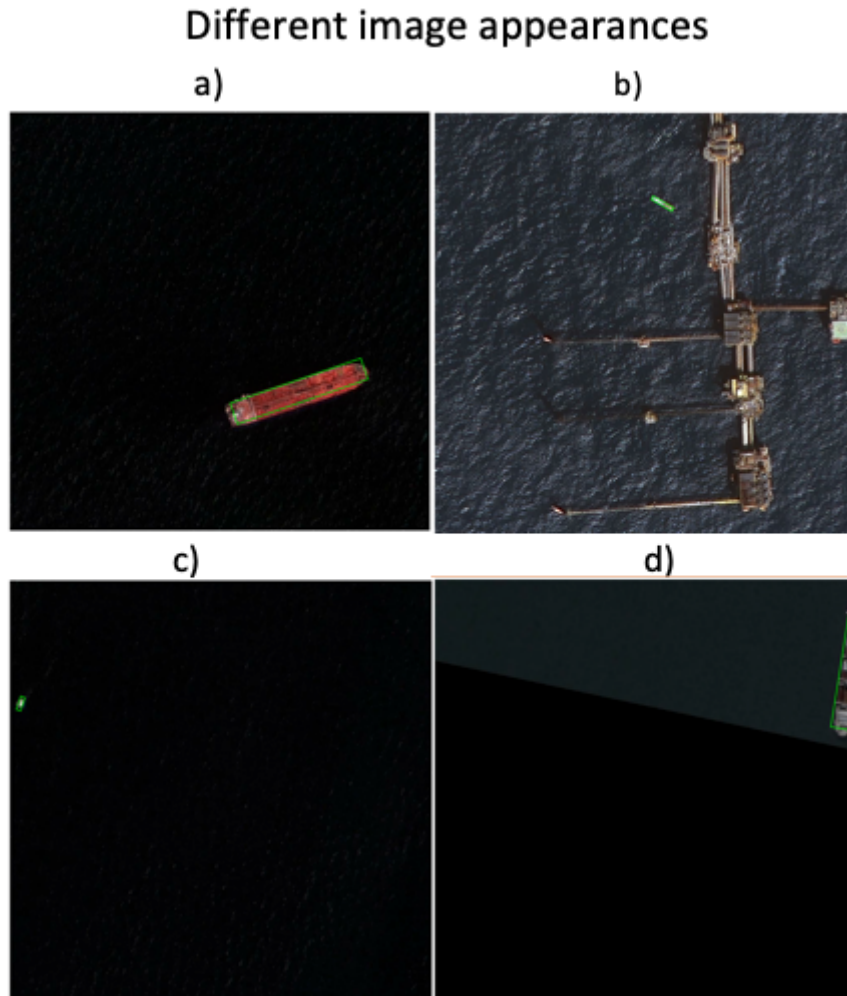


Figure 7.1: Visualization of four common SuperView appearances. The main different features are typically ship size, water texture and partial zero-padding.

Figure 7.1 exemplifies four common versions of SuperView image tiles used in the data set. Subfigures (a) and (c) display large and small ships, respectively. Subfigure (b) displays an image tile where sea waves causes the sea to be prominent. An offshore static installation is present in this image as well. Subfigure (d) displays an image tile where large proportions are zero-padded.

The data collection team at KSAT, provider of the fundamental data, clarifies that these different appearances is caused by environmental effects and that artifacts from incidence angle correction may occur. Scenes recorded with a low incidence angle may appear sharp and detailed, whereas scenes recorded with a lower incidence angle may be less sharp and detailed. When the data is this diverse, an object detector needs to be trained to operate in all such scenarios, and the importance of a large data set becomes prominent.

7.2 Data Imbalance

Class distributions must be reasonable for a deep learning model to converge properly. For pixel-wise segmentation models, this means that the proportion of positive pixels cannot be vanishingly small, but not explodingly large either. For bounding box-based models, positive annotations will learn the model how the objects are recognized and should be of sufficient scale. Uninhabited scenes will learn the detector not to produce false alarms.

For the SuperView data set annotated with RBoxes, the average area enclosed by boxes per image tile is 7100 pixels. 16500 pixels is the corresponding average area when BBoxes are used as labels. These image tiles are represented using $1024 \times 1024 \approx 10^6$ pixels, making the average ship area constituting only 0.68% and 1.6% of the total pixels when using RBoxes and BBoxes, respectively.

This data imbalance will often prevent convergence for segmentation-based models, and the optimization process needs modifications (Brenn et al., 2019). Such modifications typically include moderating the number of negative training samples, increasing the original labels, and adopting the weighted focal loss (Lin et al., 2017c). Data imbalance for grid-based models will result in a majority of empty cells in the grid. For proposal-based models, data imbalance will be reflected through the proportion of empty and rejected region proposals. For these scenarios, the imbalance is typically controlled by using carefully chosen penalty terms in the objective function, and by using a reasonable split of positive and negative training samples.

Because of the described data imbalance in the SuperView image tiles, only positive samples are included in the SuperView data set. As stated early in this chapter, the scenes under consideration are near-shore areas and the ship density is usually relatively high. It is therefore not necessary to take drastic measures. The *background class* will be trained on areas in the positive tiles where no ships are annotated. Negative tiles could have been interesting to include in the data set, with the aim of preventing false alarms. This had to be reflected in the objective function to ensure convergence. Penalty terms are included in the objective functions used for experimentation in part IV.

7.3 Augmenting Optical Satellite Images

If the data set is limited, it can be augmented using artificially generated data. If the data is used to train a deep neural network, the aim is to avoid overfitting and to increase the generalization properties (Cheng et al., 2016). The motivation for data augmentation was introduced in section 4.6.5.

Brenn et al. (2019) argue that vertical and horizontal flipping, insertion of additive Gaussian noise, rotation and scaling all are physically valid transformations for optical earth observation images. For the augmented data to be pragmatically valid, all transformations should be adopted to the sensor. This means that augmentations which are not physically valid should be dropped if it cannot be justified by increased model performance. For optical satellite data, accurate corrections for incidence angle should cause image transformations to correspond to physical differences in the sensed scene³. This means that most mappings are scenes that could have been naturally recorded by the sensor. Figure 7.2 visualizes three scenes artificially generated based on a SuperView image, supporting the allegation that these are physically valid transformations of optical earth observation images.

3. This means that a 90° rotated image using transformation will be identical to an image captured with a 90° rotated imager.

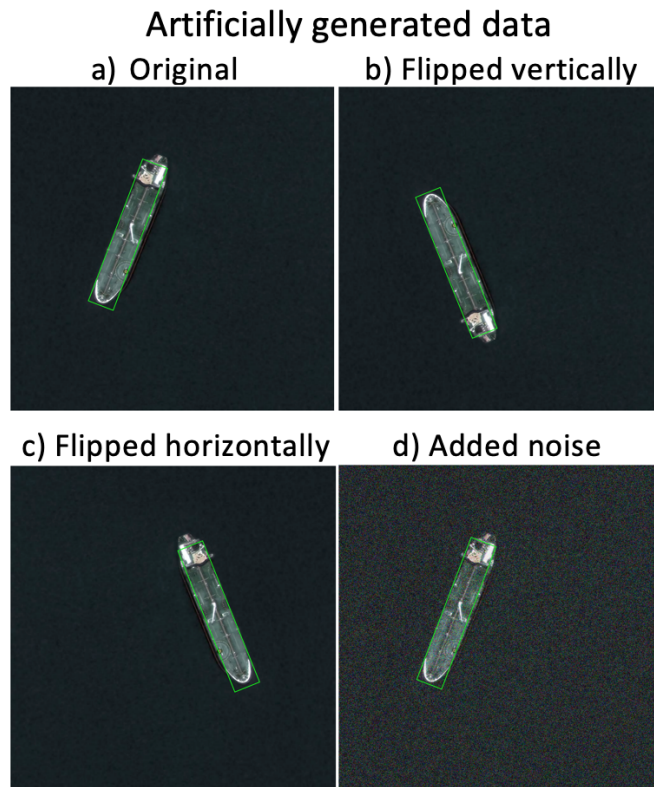


Figure 7.2: Visualization of different modes of artificially generated data based on original data to be used for data augmentation. Vertical and horizontal flipping and additive Gaussian noise are here used as example augmentation modes.

/ 8

Novelties

Three novel neural network architectures, which are extensions of existing models, will be presented in this chapter and experimented on in part IV. The aim is to extend the ODGI pipeline (Royer and Lampert, 2020) to predict RBoxes for individual object instances. This model should take advantage of object groupings to focus the evaluation on the essential areas of the image at an early stage, while describing objects in a tidier way as it adopts RBoxes. RBox predictions also lead to the beneficial result of recognizing ship headings. The author has named this model as Oriented Object Detection with Grouped Instances (OODGI).

As described in chapter 6.3.3, ODGI is composed of multiple stages of object detection. The object detectors in the different stages, $\{\phi_s\}_{s=1}^S$, will henceforth be referred to as *individual detectors*. YOLOv2 (Redmon and Farhadi, 2017) and Tiny YOLO are the individual detectors used for ODGI experimentation in this thesis. Because the novel OODGI will have a pipeline composition similar to ODGI, the individual detectors YOLOv2 and Tiny YOLO will first be extended to predict RBoxes, before they are used in the OODGI pipeline. The novel object detectors, which are extensions of YOLOv2 and Tiny YOLO, is named by the author as Oriented YOLOv2 and Oriented Tiny YOLO, respectively. These novel detectors can also be used as stand-alone object detectors.

Problems to address in the OODGI model will first be introduced. Subsequently, the novel object detectors Oriented YOLOv2 and Oriented Tiny YOLO will be presented, before the novel OODGI pipeline is presented at the end of this

chapter.

8.1 Technical Problems to Address

The problem of predicting RBoxes in the OODGI model creates a need for numerous modifications in the holistic pipeline. The overall problem can be divided into the following subproblems:

- (1) **Represent oriented bounding box:** The earlier $\theta_{BBox} = \{\theta_x, \theta_y, \theta_w, \theta_h, c\}$ representation needs to be replaced by a representation supporting orientation.
- (2) **Regress angle-related parameter:** Support for regressing the rotation-related parameter is needed¹.
- (3) **Optimize angle-related parameter:** Incorrect rotation should be penalized in the loss function. Optimization of the orientation should be included in the optimization process.
- (4) **Calculate IoU of RBoxes:** IoU calculation for RBoxes must be implemented.
- (5) **Implement data augmentation:** Data augmentation must be re-assessed and re-implemented.
- (6) **Solve and implement stage transition problem:** The stage transition in the OODGI pipeline should support RBoxes. Potential complications where group instances are rotated should be addressed.
- (7) **Constrain finalizing stage:** The last stage in the OODGI pipeline should be constrained to only output individual object instances, i.e. RBoxes.

Several of the described subproblems concern the individual detectors separately. Subproblem (1)–(5) will be addressed for the individual detectors in section 8.2 and 8.3. Subproblem (6) and (7) will be addressed when OODGI is presented in section 8.4. Subproblem (3), which initially is addressed for the individual detectors, will need modification when it becomes part of the OODGI pipeline.

1. At least, some way to infer an RBox is required. For proposal-based object detectors, this would typically mean to propose different orientations of anchors.

Because Oriented YOLOv2 and Oriented Tiny YOLO share most of the same principles and structure, *all* of the introduced subproblems are addressed equally for these object detectors. Because Oriented Tiny YOLO is considered as a lightweight version of the main Oriented YOLOv2, these problems will be described and addressed in section 8.2.

8.2 Oriented YOLOv2

The novel Oriented YOLOv2 model can be used as a stand-alone object detector to detect positioning, shape and orientation of arbitrary oriented objects, or as an individual detector in the OODGI pipeline. Oriented YOLOv2 is an extension of the base model YOLOv2 (Redmon and Farhadi, 2017). Explanation of the different subproblems to be addressed for this extension will be presented consecutively, along with the specific solution, before a summary is given.

Subproblem (1) – Represent oriented bounding box:

The earlier $\theta_{BBox} = \{\theta_x, \theta_y, \theta_w, \theta_h, c\}$ representation does not include any orientation-related parameter, and excludes the potential of describing oriented objects. In section 5.2.5, it was established that RBoxes have been successfully expressed using the six parameters: $\theta_{RBox} = \{\theta_x, \theta_y, \theta_w, \theta_l, \theta_\phi, c\}$. Here, θ_x and θ_y denote the x- and y-coordinate, respectively, while θ_w and θ_l denote the width and length, respectively. The orientation is expressed using θ_ϕ , and c denotes the predicted confidence score. The parameters θ_x , θ_y , θ_w and θ_l are normalized by division with image size, and θ_ϕ is divided by 180. The model is therefore not distinguishing the bow and stern of the ship, and all rotations are confined to $[0, 180)$. This is done similarly as in DRBox (Liu et al., 2017a), and encourages low parameter values. In section 4.6, it was stated that low parameters values should be pursued for increased generalization properties. In section 6.3.1.4, it was introduced that the *Oriented Bounding Boxes for Faster R-CNN* model (Xia et al., 2018) describes an RBox using 9 parameters. This is not adopted in this novel model because it was discussed to have too few constraints for object shape and often result in chaotic predictions.

Subproblem (2) – Regress angle-related parameter:

The grid defined for grid-based object detectors controls the positioning and size of anchors, but how can these anchors be rotated to match ground truth RBoxes? For proposal-based object detection, this typically includes proposing different oriented anchors. Initially, these grid cells are non-rotated. In this Oriented YOLOv2 model, the orientation fitting task is proposed as a direct regression task. In part IV, it is experimented on this approach, and discussed in part V whether the performance is adequate.

When solving the orientation optimization problem as a direct regression task, an additional orientation-related term in the loss function will be included. The loss function for Oriented YOLOv2 will be explained in detail when subproblem (3) is addressed in the next paragraph.

In practice, a predicted θ_ϕ parameter will be output by the CNN in addition to the original five YOLOv2 parameters listed in Table 6.1. This is simply done by expanding the earlier 5-dimensional output of the CNN to being 6-dimensional. This θ_ϕ parameter is optimized and used only by the orientation-related loss term, and for describing predictions. A simple linear activation function is applied to this sixth output component (as it will be used for regression).

Subproblem (3) – Optimize angle-related parameter:

In section 4.4.2 it was stated that regression problems are commonly solved by using a version of the MSE as loss term. For the Oriented YOLOv2 an orientation-related variant of MSE will be added to the existing YOLOv2 loss function, which already optimizes the other parameters. Leal-Taixé and Roth (2019)² argue that a simple loss term of $(\theta_\phi - \hat{\theta}_\phi)^2$ (squared angular error) is sufficient and an appropriate loss term to optimize rotations.

Based on this, the proposed Oriented YOLOv2 loss function, given in equation 8.1, includes a sum of squared errors (a variant of MSE, expressed in equation 4.10) loss term for regressing the orientation parameter θ_ϕ , where $\theta_\phi \in [0, 1]$.

$$\begin{aligned} L_{\text{Oriented YOLOv2}} &= L_{\text{YOLOv2}} + L_{\text{orientation}} \\ &= L_{\text{YOLOv2}} + \lambda_{\text{rotation}} \sum_{i=0}^{C^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[\left(\theta_\phi^{(i)} - \hat{\theta}_\phi^{(i)} \right)^2 \right] \end{aligned} \quad (8.1)$$

The L_{YOLOv2} term in equation 8.1 is the YOLOv2 loss expressed back in equation 6.11, $\lambda_{\text{rotation}}$ is a hyperparameter regulating the weighting of the rotation loss term relative to the total loss, $\theta_\phi^{(i)}$ and $\hat{\theta}_\phi^{(i)}$ are the normalized true and predicted rotation of the RBox, respectively. Remaining notation is inherited from equation 6.11. It is not convenient to output a transformed $\hat{\theta}_\phi$ from the CNN (like the mappings described in equation 6.10), since orientation is not cell-dependent.

This additional loss term is processed similarly as the other loss terms during the optimization process.

2. The paper of Leal-Taixé and Roth (2019) concerns an oriented version of the YOLOv3 (Redmon and Farhadi, 2018) model, which is applied to 3-dimensional objects.

Subproblem (4) – Calculate IoU of RBoxes:

In the ordinary YOLOv2 process, IoU between BBoxes is applied three times: to select the prior BBox responsible for a ground truth object during training (i.e., the binary $\mathbb{1}_{ij}^{obj}$ and $\mathbb{1}_{ij}^{noobj}$), for NMS during inference, and to evaluate mAP during validation. Calculating the exact IoUs between RBoxes is tedious and computer-intensive, and is not feasible for near real-time processing.

Approximated calculations of IoU between RBoxes are used for example in R²PN (Zhang et al., 2018) and DRBox (Liu et al., 2017a). However, the training and inference time still increases when using such calculations. The approximation in DRBox (Liu et al., 2017a) is based on rotating one of the RBoxes to match the angle of the other RBox and calculate the IoU in this translated coordinate system. Because the IoU criterion is also calculated during inference, it is desirable to use a fast computation. Experiments on DRBox in part IV will indicate that this approach is not feasible for near real-time processing.

To address the problem of IoU calculation to find the prior RBox that is considered responsible for predicting an object, the RBoxes are transformed to a corresponding BBox and the standard IoU is calculated on these BBoxes. This approach is chosen in lack of better alternatives. In section 6.3.2, it was stated that the responsible prediction also has the best fitting shape and aspect ratio. When calculating IoU between RBoxes using this proposed approach, it will not cause the model to make better predicted rotations for every iteration in the same way that shape and aspect ratio will be improved. The rotation parameter will be learned by using regression, as stated earlier, which may cause a higher variation in parameter values and may result in poorer predictions. When transforming to BBoxes, the responsible RBox will now be the one having the best matching BBox with the ground truth BBox.

The problem of IoU calculation between RBoxes in the NMS step during inference, is addressed using an identical approach as the one described in previous paragraph. It is conceivable that this is a drawback when processing scenes with high object density. This thesis concerns grouped objects, and it may therefore be worth noting this when approaching the experiments in part IV.

IoU calculation between predicted RBoxes for mAP evaluation during validation is performed similarly as was described in the previous two paragraphs. Because this is performed only during validation, the argument of avoiding algorithms that is tedious and computer-intensive is not that pertinent.

When ignoring this argument, the calculation of IoU between RBoxes becomes a simple problem³. However, such an approach has not been adopted because it is desirable to produce mAP results that are directly comparable to other models.

Subproblem (5) – Implement data augmentation:

In section 7.3, it was stressed that selection of mappings for data augmentation should consider the specific data and instrument. Augmentations are desired to be physically valid, meaning that one should not be able to detect manipulation of the data based on knowledge of the nature of the sensor and data. In other words, the sensor should be able to produce data realizations that are identical to the augmentations. When changing the bounding box format, i.e. adopting the $\theta_{RBox} = \{\theta_x, \theta_y, \theta_w, \theta_l, \theta_\phi, c\}$ representation, the data augmentation modes need to be reconsidered. A reimplementations of the augmentation mappings are also necessary.

In section 7.3, five transformations were specifically mentioned: vertical and horizontal flipping, addition of Gaussian noise, rotation and scaling. Because the optical satellite images remain the same, and the only modification is the annotation format, all these mappings would entail physical correctness.

Summary

Except for the mentioned modifications, the Oriented YOLOv2 architecture is similar to the standard YOLOv2 (Redmon and Farhadi, 2017) architecture. Other than having a 6-dimensional output, as described in this chapter, the CNN is identical to the one presented in Table 6.1. The same concepts as presented in section 6.3.2.2 apply.

In addition to the previous hyperparameters applicable for the standard YOLOv2, an additional $\lambda_{rotation}$ hyperparameter concerning the weighting of the orientation loss term in the total loss function is introduced. This Oriented YOLOv2 model is prepared to make adjustments in the grid cell size according to a user-defined image tile size. The grid cell size will be 32 times smaller than the size of the image tile. I.e., an image tile of 1024×1024 pixels will result in a grid cell size of 32×32 pixels, 512×512 pixel tiles result in a grid cell size of 16×16 pixels, etc. This allows for experimentation on the subject of speed versus accuracy trade-off.

3. An approach for exact IoU calculation between RBoxes is for instance proposed at: <https://stackoverflow.com/questions/44797713/calculate-the-area-of-intersection-of-two-rotated-rectangles-in-python>.

8.3 Oriented Tiny YOLO

As the Oriented YOLOv2 is already presented, it is elementary to present the Oriented Tiny YOLO. Recall from section 6.3.2.3 that the Tiny YOLO model is identical to the YOLOv2 model, except practicing a more lightweight CNN. The novel Oriented Tiny YOLO is, similarly as for the non-oriented versions, a direct replica of the presented Oriented YOLO model, except using the identical lightweight CNN as Tiny YOLO (described in Table 6.2), but now with a 6-dimensional output.

When the Oriented Tiny YOLO model now is introduced, diverse complexities of the OODGI pipeline may be experimented on to study its speed versus accuracy properties, and to select a combination that is customized for the specific problem. Recall that Oriented YOLOv2 and Oriented Tiny YOLO are presented as stand-alone object detectors. Now that two novel versions of stand-alone oriented object detectors are introduced, both receptive for diverse grid cell sizes, the speed versus accuracy trade off can be experimented on for stand alone detectors as well.

8.4 Oriented Object Detection with Grouped Instances

The novel OODGI pipeline with S stages $\{\phi_s\}_{s=1}^S$ will now be introduced. Subproblem (6) and (7) will be addressed in succession, before the overall model is summarized. Subproblem (3), initially addressed in section 8.2, will need modifications when used in this pipeline. Each stage ϕ_s consist of one oriented individual detector.

Recall Figure 6.3, presenting the ODGI pipeline. This figure is highly relevant for the OODGI model as well. All stages are linked, $\phi_s \circ \dots \circ \phi_1$, $S > 1$, and each stage is capable of predicting both individual and grouped object instances. Candidates for further refinement predicted at stage ϕ_s will be the input to the consecutive stage ϕ_{s+1} . Recall from section 6.3.3 that a crop denotes an image segment that is a candidate for further refinement in the consecutive stage, and can contain both individual and group instances, meeting certain criteria.

The scheme for sorting predictions at a stage ϕ_s (which is not the finalizing stage, $s < S$) is, as introduced in section 6.3.3, summarized as:

- i) $c \leq \tau_{low}$: The prediction is discarded.
- ii) $c \geq \tau_{high}$ and $g = 0$: The predictions is considered as an individual object instance and skips consecutive stages.

iii) ($c > \tau_{low}$ and $g = 1$) or ($\tau_{low} \leq c < \tau_{high}$ and $g = 0$): Prediction is either a group or an individual with medium confidence and is passed on to the consecutive stage.

Here, τ_{low} and τ_{high} are the user-defined weak and strong confidence thresholds, respectively, c is the predicted confidence score, and g is the binary group flag, all described in detail in section 6.3.3.

Subproblem (6) – Solve and implement stage transition problem:

Because a predicted group object instance is passed to the consecutive stage for refinement, it is treated as an image in the consecutive stage and poses problems if such a group object instance is rotated. The proposed solution is to predict RBoxes for individual object instances, and BBoxes for group object instances. The original model does already support division of individual and group instances. It does this by predicting the binary group flag g , as was introduced in section 6.3.3. Here, this distinction is encouraged by labeling group object instances using BBoxes and individual object instances using RBoxes. Figure 8.1 exemplifies what these mixed labels might look like. Similarly as for ODGI, described in section 6.3.3, individual object instances occurring in at least one common cell are considered part of a grouped object instance. In OODGI, a traditional BBox fully enclosing all individual object instances (annotated as RBoxes) in this group is formed. The group instance BBoxes will follow the $\{\theta_x, \theta_y, \theta_w, \theta_h, \theta_\phi, c\}$ representation for format consistency, but with $\theta_\phi = 0$.

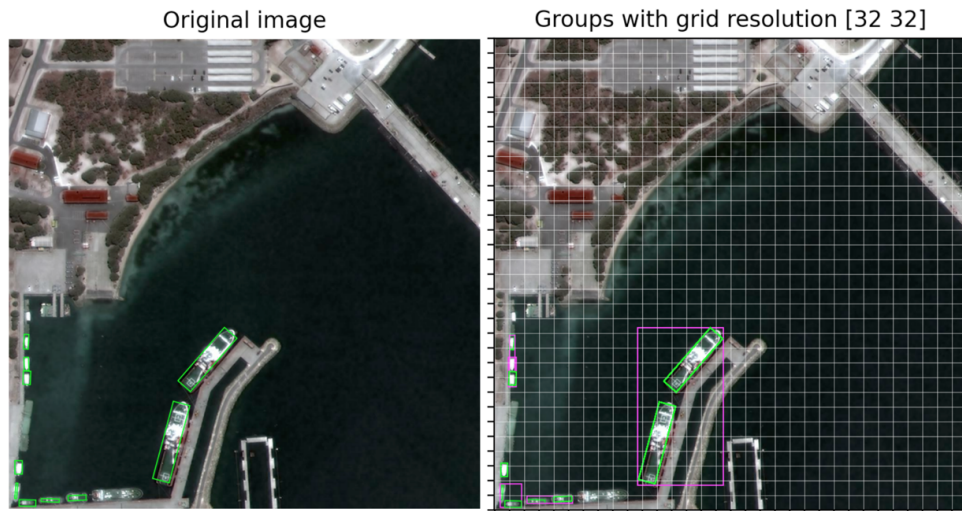


Figure 8.1: Visualization of how a group mask (purple) is formed based on individual instance RBoxes (green). The group mask is always a BBox fully enclosing all RBoxes that are considered a member of this group. One ship is not annotated as a result of deficiencies in the labeling process.

During training, the model will learn that grouped instances have zero rotation and that individual instances allow rotation. Deviations will occur, especially in the early training process. Recall from section 6.3.3, that predictions at intermediate stages fulfilling ($c > \tau_{low}$ and $g = 1$) or ($\tau_{low} \leq c < \tau_{high}$ and $g = 0$), meaning that the prediction is either a group or an individual with medium confidence, will be passed on to consecutive stages for refinement. Hence, not only predictions considered as group instances are refined in consecutive stages. These two mentioned scenarios are the origin behind the following constraint: If a candidate for further refinement has nonzero rotation $\theta_\phi \neq 0$, it is transformed to a corresponding BBox before being input to the consecutive stage. For predictions that are considered individuals with medium confidence, this will be a specifically recurring event. Individual object instances are annotated using RBoxes, and the individual detectors will constantly try to predict a correct orientation, even at early stages. However, because the ground truth object annotation will be mapped to the crop-relative coordinate system (as will be described in the following paragraph), the consecutive stage will still be able to recognize the object characteristics.

Originally, all annotations are relative to the complete image tile. However, when crops are formed and passed on to consecutive stages in the pipeline, the annotations should be recalculated to be relative to this respective crop. The simple trigonometry rules, mathematically expressed by equation 8.2, will do the mapping $\{\theta_x, \theta_y, \theta_w, \theta_l, \theta_\phi\} \rightarrow \{\theta_{x_c}, \theta_{y_c}, \theta_{w_c}, \theta_{l_c}, \theta_{\phi_c}\}$ to a new crop-relative coordinate system. In equation 8.2, $\theta_{crop_min_x}$ and $\theta_{crop_min_y}$ denotes the minimum x- and y- coordinates of the crop⁴, respectively, and θ_{crop_w} and θ_{crop_h} denotes the width and height of the crop, respectively. Implementation of this mapping is appended in section 14.2.

$$\begin{aligned}
 \theta_{x_c} &= \theta_x - \theta_{crop_min_x} \\
 \theta_{y_c} &= \theta_y - \theta_{crop_min_y} \\
 \theta_{w_c} &= \frac{\theta_w(\sin \theta_\phi + \cos \theta_\phi)}{\theta_{crop_w}} \\
 \theta_{l_c} &= \frac{\theta_l(\sin \theta_\phi + \cos \theta_\phi)}{\theta_{crop_h}} \\
 \theta_{\phi_c} &= \theta_\phi
 \end{aligned} \tag{8.2}$$

4. It is also common to parametrize a BBox using $\{x_{min}, y_{min}, x_{max}, y_{max}\}$. Then $\theta_{crop_min_x}$ and $\theta_{crop_min_y}$ will correspond to x_{min} and y_{min} , respectively.

Subproblem (7) – Constrain finalizing stage:

The concluding stage of the OODGI is, similarly as for ODGI, constrained to only output individual objects. This constrain is included as it is desirable to produce results that are comparable with other object detectors. If enough stages are included relative to the task under consideration, all such predictions should allow orientation and be RBox predictions.

Because RBox annotation is still used for individual object instances, the model will learn to output oriented predictions for such scenarios during optimization. If the number of stages S is not sufficient, non-rotated predictions may occur at the concluding stage. This is an artifact of the choice of S , and is not considered a problem. All predictions at the concluding stage are transformed back to the original image coordinate system. This is done by solving the respective expressions in equation 8.2 for $\{\theta_x, \theta_y, \theta_w, \theta_l, \theta_\phi\}$, over all S stages, $\phi_S \circ \dots \circ \phi_1$, $S > 1$.

The problem of IoU calculation has already been addressed, and the same principles applies for this model. However, IoU calculation (subproblem (4)) is now performed with yet another purpose: for NMS at intermediate stages to select the best crops to be refined at consecutive stages. Recalling that these crops are basic traditional BBoxes, this is simply executed as a standard IoU calculation.

Subproblem (3) – Optimize angle-related parameter:

Subproblem (3) is resumed because the particular loss function presented for Oriented YOLOv2 and Oriented Tiny YOLO is not applicable for OODGI. Recall from the L_{ODGI} loss in equation 6.12 that it is composed of the terms L_{groups} , L_{coords} and $L_{offsets}$. L_{ODGI} has proven to be well-functioning, thus it is desirable that the new L_{OODGI} loss is based on the same principles. L_{groups} needs no modification as the groups are identical as before. The labels of grouped object instances are also identical as earlier. L_{coords} needs modification to optimize the angle-related parameters. The L_{coords} term consist of three terms whereas only the first term optimizes prediction of the earlier $\{\theta_x, \theta_y, \theta_w, \theta_h\}$ BBox parameters. As these mentioned parameters are optimized using a least square error loss, optimization of the additional angle-related parameter is also performed using a least square error loss version. $L_{offsets}$ concerns crop offsets at intermediate stages. It is described earlier in this chapter that crops at intermediate stages will behave equivalently as earlier, thus it is not necessary to modify $L_{offsets}$.

The loss function used to optimize OODGI, mathematically described by equation 8.3, is a combination of the ODGI loss and the Oriented YOLOv2 loss. Here, L_{ODGI} is the total ODGI loss, expressed in equation 6.12, and $L_{orientation}$ is the rotation-related loss term introduced for Oriented YOLOv2 in section 8.2,

expressed in equation 8.1.

$$L_{OODGI} = L_{ODGI} + L_{orientation} \quad (8.3)$$

Summary:

The novel OODGI is relatively similar to the baseline ODGI model, with the described modifications. Non-described concepts can be assumed to be equally applied as for the ODGI model. The overall loss function, presented in equation 8.3, introduces an additional $\lambda_{rotation}$ hyperparameter weighting the $L_{orientation}$ term in the total loss function.

The total set of hyperparameters consist of τ_{low} , τ_{high} , τ_{nms} , τ_s , S , λ_{conf} , λ_{noobj} , $\lambda_{rotation}$ and δ . Recall from the ODGI introduction in section 6.3.3 that τ_{low} and τ_{high} are the weak (τ_{low}) and strong (τ_{high}) confidence thresholds, τ_{nms} is the NMS threshold, τ_s is the number of crops to evaluate at stage ϕ_s and S is the total number of stages in the pipeline. The hyperparameters λ_{conf} , λ_{noobj} and $\lambda_{rotation}$ are for weighting loss terms, and δ is a margin hyperparameter regularizing the minimum desirable overlap between a prediction and the corresponding BBox. Suggested hyperparameters and training scheme is described among the experimental setup in section 10.1.

When this architecture now is introduced as a pipeline with an unlimited number of stages S , it is considered a general model that can be customized to fit various problems. Because each stage is capable of predicting either an individual instance or to propose a grouped instance for further analysis in consecutive stages, OODGI can be considered a hybrid of grid-based and proposal-based object detector, where each stage that can propose grouped instances is a RPN.

Royer and Lampert (2020) argue that $S = 2$ stages are sufficient for most ordinary remote sensing problems, and that $S > 2$ is more suited for very-large scale scenarios, e.g. gigapixel images. This argumentation is applicable to the novel OODGI as well. Because most crops are of limited resolution, a suggestion of using more lightweight individual detectors at intermediate stages to increase inference speed is appropriate. It is rarely desirable to use an exaggerated individual detector to evaluate a crop of limited resolution.

Part IV

Experiments and Experimental Setup

This part concerns all experiments performed on selected models in this thesis. Experimental setups are described, and experimental results are presented, both for reference and novel detectors.

Articles on object detection usually present experimental results in the form of inference time, mAP and predicted examples. However, false alarms, misses and proportion of correct predictions are also interesting measures during model evaluation in the field of ship detection, and will be studied in this part. Chapter 9 discloses experiments performed on reference models, including experimental setup, inference time, AP and example predictions. Chapter 10 discloses experiments performed on the novel methods presented in chapter 8, including experimental setup, inference time, AP and example predictions. An accuracy analysis on main models is presented in chapter 11. Some reported average precisions in Table 10.1 may be considered as unforeseen and is the origin behind a model stochasticity analysis, also presented in chapter 11. Experimental results are presented in this part, but are discussed in detail in part V.

When example predictions are visualized in this part, basic ships⁵, larger ships, areas of high ship density and harbor scenes are scenarios that will be particularly studied. These mentioned scenarios are together considered to constitute a holistic perspective of potential scenarios. All experiments throughout this thesis are calculated on a GeForce RTX 2080 Ti 11 GB GPU⁶, during training, testing and inference.

5. A basic ship refers to the most common appearance of a ship: a smaller ship with no neighboring ships and no significant difficulties.

6. Additional specifications is found at <https://www.nvidia.com/nb-no/geforce/graphics-cards/rtx-2080/>.

/9

Experiments on Reference Models

Numerous architectures for object detection were reviewed in chapter 6. As introduced in part I, the scope of this thesis is to study the properties of using grouped object instances and rotatable annotations for increased precision and a more prominent representation. Concretely, ODGI (Royer and Lampert, 2020) is the model that is based on grouped object instances which will be reviewed in this thesis. In chapter 10, the novel extension OODGI, which predict object orientations, will be experimented on. ODGI is built upon YOLOv2 (Redmon and Farhadi, 2017) and Tiny YOLO. Thus, these are also included for experimentation on ODGI improvement from the baseline models. DRBox (Liu et al., 2017a) is experimented on to give insight into performance of preceding models using RBoxes. Faster R-CNN (Ren et al., 2015) is included in the experiments to give a comparison with a state-of-the-art proposal-based object detector.

9.1 Experimental Setup

During inference and validation on all object detectors experimented on in this chapter, the hyperparameters τ_{IoU} and τ_c are included, and properly determined values are desired to achieve satisfactory results. τ_{IoU} and τ_c are the IoU threshold and the confidence score threshold, respectively, but applied during NMS. Using $\tau_{IoU} = 0.5$ is a standard choice, recurring in object detection experiments (Royer and Lampert, 2020). During NMS, this

will suppress predictions having $IoU \geq 0.5$ with already accepted predictions¹. $\tau_{IoU} = 0.5$ is used for experiments on reference models in this chapter for easier comparison to other models.

The τ_c hyperparameter is the threshold that rejects all predictions having a confidence score $c < \tau_c$. How this hyperparameter should be determined depends on the data set complexity and the model properties. A complex and computer-intensive model will typically be more confident in each prediction (higher c) than an operational real-time model (Ren et al., 2015). A common approach is to test for different adjustments of τ_c and inspect how the mAP unfolds. Based on unreported experiments on the SuperView data set, and experimental results in Liu et al. (2017a), Redmon and Farhadi (2017) and Royer and Lampert (2020), $\tau_c = 0.4$ appears to be an appropriate confidence score threshold for the grid-based models. This is used throughout experiments on grid-based architectures. The confidence score threshold τ_c for the proposal-based Faster R-CNN is introduced in the subsequent section. The grid-based models are operating in near real-time and often have a low confidence score on small and cluttered objects. A relatively low τ_c will be more receptive to such objects, but may also cause a higher false alarm rate.

To compensate for the data imbalance issue (section 7.2), only positive image tiles are used in the training data set, i.e. images that have at least one annotated object. The Faster R-CNN² and DRBox³ implementations used in the following experiments are written in the Keras framework (Chollet et al., 2015). Implementations of YOLOv2, Tiny YOLO and ODGI⁴ are written in Tensorflow (Abadi et al., 2015).

9.1.1 Faster R-CNN:

The Faster R-CNN model is trained using $k = 9$ region proposals for each sliding window in the RPN, similarly as for the official experiment (Ren et al., 2015). These $k = 9$ region proposals include the scales $\{64, 128, 256\}$ and the aspect ratios $\{1 : 1, 1 : 2, 2 : 1\}$. The aspect ratios are adopted from those suggested by Ren et al. (2015). The scales are halved with respected to the suggested values, because the SuperView data set is of larger scale than the multi-object category images used in the original experiments.

1. Already accepted predictions have a higher confidence score, as stressed in section 5.2.1.
2. Implementation of the Faster R-CNN is based on the open source code:
https://github.com/RockyXu66/Faster_RCNN_for_Open_Images_Dataset_Keras.
3. Implementation of the DRBox is based on the open source code:
https://github.com/ppontisso/DRBox_keras.
4. Implementations of these models are based on the open source code:
<https://github.com/ameroyer/ODGI>.

The data is augmented with a probability of 0.7 for doing a transformation. If an image is selected to be transformed, it is mapped using one of the equiprobable modes: horizontal flip, vertical flip or 90° rotation. The model is trained using the Adam optimizer (Kingma and Ba, 2014) with a learning rate of $\mu = 10^{-5}$ on classifier and RPN separately (adopted from original source code). Additional hyperparameters are set according to default values, stated in section 4.5.2.

Based on the original source code, VGG (Simonyan and Zisserman, 2014) is used as the CNN for feature extraction, allowing the use of pretrained weights. It is therefore initialized using weights pretrained on Image Net Large Scale Visual Recognition (ILSVRC) (Russakovsky et al., 2015). Recall from section 6.3.1.2, the CNN is shared between the RPN and the RoI pooling layer. This will lead to faster convergence during training, and potentially better precision.

Based on unreported experiments, a confidence score threshold $\tau_c = 0.6$ has proven to be appropriate for this problem, and is used during experimentation. Dropout (Hinton et al., 2012b) is included in the *cls* classification layer for increased generalization properties (based on original source code). Similar activation functions as in the original Faster R-CNN architecture is used (Ren et al., 2015). ReLU activation functions are used at intermediate layers. Out of the *cls* classification layer, the multi-class version of sigmoid $\sigma(x)$, the *softmax* activation function, is used. This is a very common practice, allowing the class scores to be interpreted as probabilities. For the *reg* regression layer, a simple linear activation function is used. The straightforward Faster R-CNN is used, thus all ships are labeled using traditional BBoxes.

9.1.2 DRBox

As suggested in Liu et al. (2017a), the DRBox model is trained using 30 variations of prior RBoxes at each cell location in the grid. These 30 variations are the predefined shapes: $\{20 \times 8, 40 \times 14, 60 \times 17, 80 \times 20, 100 \times 25\}$ pixels, and the angles: $\{0, 30, 60, \dots, 150\}$ degrees, all inspired by the original source code.

This architecture does also apply a VGG (Simonyan and Zisserman, 2014) model as the CNN for feature extraction. It is therefore initialized with weights pretrained on ILSVRC (Russakovsky et al., 2015) for a faster convergence, and potentially to achieve better precision. Because the data set is of limited extent, it is augmented with a probability of 0.5 for augmenting each image using one of two equiprobable sequences: Sequence one will do a horizontal flip, a vertical flip, random rotation and zoom in, all with an individual probability of 0.5. Sequence two will do a horizontal flip, a vertical flip, random rotation and zoom out, again with an individual probability of 0.5.

Based on original architecture and source code, optimization is performed using the Adam optimizer (Kingma and Ba, 2014) with a learning rate of $\mu = 10^{-5}$. Additional hyperparameters are determined according to default values, stated in section 4.5.2. The L_2 norm regularization strategy is added to the objective function for penalizing large parameter values, and hence potentially increasing the generalization properties.

Activation functions are adopted from the original architecture. ReLU activation functions are used at intermediate layers. The sigmoid activation function $\sigma(x)$ is applied to the confidence scores. The sigmoid activation function confines the confidence score to $c \in [0, 1]$, allowing for c to be interpreted as probabilities. Simple linear activation functions are applied to the predicted box parameters. All objects are labeled using RBoxes. This model is quite computer-intensive, and was restricted by the available memory budget during training. The result was to use `batch_size = 2`. This may cause the different batches to be more variable.

9.1.3 YOLOv2

Redmon and Farhadi (2017) presented the YOLOv2 approach as a detection followed by a classification problem. In section 6.3.2.2, the architecture was presented as a detection problem, and the classification branch was omitted. This is because the ship detection experiments performed in this thesis are single-object category problems, and classification is hence irrelevant.

Each image tile has a probability of 0.7 to be transformed. This augments the data to a great extent based on the limited original data set. If an image tile is selected to be transformed, it is mapped using one of the modes: horizontal flip, vertical flip, and additive Gaussian noise, each equally probable. Based on original source code, the YOLOv2 model is trained using the Adam optimizer (Kingma and Ba, 2014) with a learning rate of $\mu = 10^{-5}$, and default parameters beside this. No pretrained weights are applied. Instead, the He initialization (He et al., 2015) is used for weight initialization. He initialization is inspired from original source code, and is common to use in conjunction with ReLU activation functions (or Leaky ReLU). Leaky ReLU activation functions⁵ are used at intermediate stages, similarly is in the original source code. The sigmoid activation function $\sigma(x)$ is applied to the confidence scores, allowing for probability interpretation.

Based on original source code, several regularization strategies are applied to achieve better generalization properties. Batch normalization (Ioffe and Szegedy, 2015) is applied to intermediate convolutional layers. The L_2 norm is

5. Leaky ReLU is an offspring from the standard ReLU, where negative values are mapped linearly to a small number different from zero.

added to the objective function for penalizing large parameter values.

YOLOv2 experiments are performed on two architecture variations: YOLOv2 1024 and YOLOv2 512. These denotes grid cell sizes that correspond to image sizes of 1024×1024 pixels and 512×512 pixels⁶, respectively. Because the grid cell size corresponds to a downsampling factor of 32, YOLOv2 1024 and YOLOv2 512 operate with 32×32 and 16×16 pixels per grid cell⁷, respectively. Recall that the cells in this grid can be interpreted as prior BBoxes. Thus, decreasing the grid cell size does also decrease the model complexity. These models are selected for experimentation because they are considered to have the highest potential for producing adequate and interesting results.

In section 6.3.2.2, it was stated that $\lambda_{coord} = 1$ and $\lambda_{noobj} = 5$ are suggested hyperparameters for the normal remote sensing data set (Royer and Lampert, 2020). These values are used in these experiments.

9.1.4 Tiny YOLO

Experiments on Tiny YOLO are conducted identically as for YOLOv2, but now using the lightweight Tiny YOLO network. The same grid cell sizes as described in section 9.1.3 apply. However, the extra lightweight version Tiny YOLO 512 is omitted, as it is assumed to be too elementary for the SuperView data set.

9.1.5 ODGI

Royer and Lampert (2020) argued that $S = 2$ stages is sufficient for the ordinary remote sensing data set. As the SuperView data set is considered an ordinary remote sensing data set, $S = 2$ stages are used for experimentation in this chapter. Different variants of the ODGI model are experimented on, with the intention of analyzing its speed versus accuracy properties. This involves different combinations of individual detectors in stage one ϕ_1 and stage two ϕ_2 , and different grid resolutions in the corresponding stages. YOLOv2 and Tiny YOLO are the individual detectors used for experimentation. The ODGI pipeline will be described by the individual detectors at the respective stages in the following. An ODGI pipeline using YOLOv2 in stage one ϕ_1 and Tiny YOLO in stage two ϕ_2 will hence be denoted as ODGI yt. The grid cell sizes used for experiments on YOLOv2 and Tiny YOLO, as described in section 9.1.3, are used for experimenting on ODGI as well. Hence, the ODGI yt 512-256 model denotes a pipeline with YOLOv2 and grid cell size of 16 in ϕ_1 , and Tiny YOLO and grid cell size of 8 in ϕ_2 .

6. Image tiles of size 1024×1024 , 512×512 and 256×256 pixels will hereby be appended to the model name as labels 1024, 512 and 256, respectively.

7. Grid cell sizes of 32×32 , 16×16 and 8×8 pixels per grid cell will hereby be denoted as 32, 16 and 8 cells, respectively.

Augmentation, regularization and activation functions are applied likewise as for YOLOv2 and Tiny YOLO, described in section 9.1.3. Similarly as in Royer and Lampert (2020), Adam (Kingma and Ba, 2014) is used for optimization with respect to the loss function, expressed by equation 6.12. Default values are used for the Adam optimizer.

Royer and Lampert (2020) performed some experiments on different combinations of hyperparameters. Recall from section 6.3.3 that the total set of hyperparameters in ODGI includes τ_{low} , τ_{high} , τ_{nms} , τ_s , S , λ_{conf} , λ_{noobj} and δ , where τ_s is reduced to τ_1 when using $S = 2$ stages. The thresholds τ_{low} and τ_{high} are the weak (τ_{low}) and strong (τ_{high}) confidence thresholds, τ_{nms} is the NMS threshold, τ_s is the number of crops to evaluate at stage ϕ_s and S is the total number of stages in the pipeline. The hyperparameters λ_{conf} and λ_{noobj} are for weighting loss terms, and δ is a margin hyperparameter regularizing the minimum desirable overlap between a prediction and the corresponding BBox. Diverse τ_{low} , τ_{high} , τ_{nms} and τ_1 are practiced during training and inference. This sets few constraints on the abilities to learn during training, while restricting the process time during inference. Based on the success of using $\lambda_{conf} = 5$, $\lambda_{noobj} = 1$ and $\delta = 0.0025$ in experiments performed by Royer and Lampert (2020), the same loss penalty terms are applied during experiments on the SuperView data set, because these data sets are considered to be relatively similar.

To let as many predictions flow through the training process as possible, no thresholding is used, i.e. $\tau_{low}^{train} = 0$, $\tau_{high}^{train} = 1$, and $\tau_{nms}^{train} = 1$. The maximum number of crops to output from ϕ_1 during training, τ_1^{train} , is adjusted according to the maximum number that is allowed by the memory budget. For the hardware used in these experiments, this corresponds to $\tau_1^{train} = 10$.

Based on hyperparameter experimentation, Royer and Lampert (2020) suggested $\tau_{low}^{test} \in \{0, 0.1\}$, $\tau_{high}^{test} \in \{0.6, 0.7\}$, $\tau_{nms}^{test} = 0.25$ and $\tau_1^{test} = 6$, dependent on the data set complexity and the speed versus accuracy weighting⁸. Based on the aforementioned suggested values and on unreported experiments performed on the SuperView data, $\tau_{low}^{test} = 0.05$, $\tau_{high}^{test} = 0.6$, $\tau_{nms}^{test} = 0.5$ and $\tau_1^{test} = 8$ are used during all ODGI experiments in this thesis. $\tau_{nms}^{test} = 0.5$ is used for easier comparison to other models. $\tau_1^{test} = 8$ is determined based on a worst case scenario: as harbor areas are of special interest in this grouped instance analysis, scenes with numerous grouped instances, like the one visualized in Figure 1.1, is a repetitive scenario.

8. Royer and Lampert (2020) actually experimented on two data sets of different complexity. However, the SuperView data set used for experimentation in this thesis is most comparable to the data set of highest complexity, and is the one that will be referred to.

Each stage $\{\phi_s\}_{s=1}^2$ is trained separately, meaning that the loss of ϕ_2 will not penalize ϕ_1 and vice versa. The input to ϕ_2 is the result from ϕ_1 . Royer and Lampert (2020) argue that it is cumbersome to wait for one stage to be completely trained before training the consecutive stage, and that ϕ_1 produces results that are useful for training ϕ_2 after only a few training epochs. A training procedure that involves training ϕ_1 for n_e epochs before including ϕ_2 in the training process is thereby proposed. Even though $n_e = 3$ is suggested in Royer and Lampert (2020), $n_e = 30$ is used when training ODGI on the SuperView data set because of the few data samples in this data set.

Experiments are performed on the ODGI yt 1024-512, ODGI yt 512-256 and ODGI tt 1024-512 models, as these are considered the most relevant compositions with the highest potential for producing adequate results on the relatively complex SuperView data set.

9.2 Experimental Results

In this section, experimental results on the setups described in previous section is presented. Inference time, AP and the number of parameters (network weights) in the different setups are presented in Table 9.1. Example predictions performed by the Faster R-CNN and the DRBox models are visualized in Figure 9.1 and 9.2, respectively. Scenes of higher object density is not represented by figure 9.2 as no such images was included in the used inference data set. Recall that scenes of special interest in this analysis are basic ships, larger ships, areas of high object density and harbor scenes. Figure 9.3 and 9.4 visualizes example scenes of a harbor and an area of high object density, respectively. Predicted results produced by the YOLOv2 1024 and Tiny YOLO models are included in these figures. These scenarios proved to be the most challenging scenarios, and other scenarios are considered to be more elementary and thus less interesting. Predicted figures by the more lightweight YOLOv2 512 model is omitted as it is assumed to be too elementary for these complex scenes, and due to the failing results reported in Table 9.1.

Particularly noteworthy details from Table 9.1 are the large inference time for DRBox, and the improvement in precision of ODGI compared to the baseline models. A noteworthy detail in Figure 9.1 is that smaller ships are not assigned a small enough BBox. Because most ships in harbor areas are difficult to detect, they are typically predicted with a low confidence threshold. When generating Figure 9.3, a lower confidence score threshold τ_c is used to achieve a more explanatory figure.

Table 9.1: Results of experiments with the SuperView data set described in chapter 7 on different reference models with the experimental setups described in section 9.1. Inference time are obtained with a GeForce RTX 2080 Ti GPU.

Model	Inference time [s]	AP	# parameters
Faster R-CNN	0.373	0.631	137 M
DRBox	2.364	0.579	27 M
YOLOv2 1024	0.116	0.427	51 M
YOLOv2 512	0.042	0.321	51 M
Tiny YOLO 1024	0.103	0.425	11 M
ODGI yt 1024-512	0.126	0.602	62 M
ODGI yt 512-256	0.045	0.462	62 M
ODGI tt 1024-512	0.116	0.595	22 M

Annotated ships predicted using Faster R-CNN

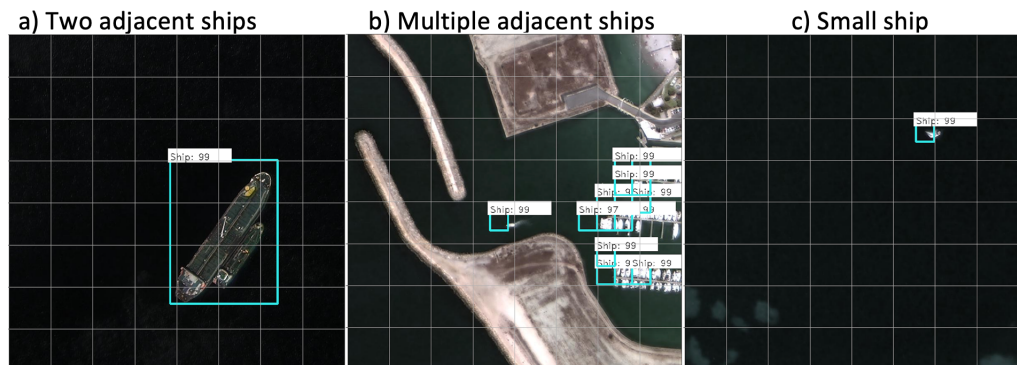


Figure 9.1: Visualization of Faster R-CNN results for three different scenarios. Scene a) consists of two easily detectable adjacent ships. Because the ships are so close and traditional BBox are used, additional predictions were removed during NMS. Scene b) is a harbor with numerous smaller ships. The model fails to describe all ships in the scene because of overlaps, and the result is messy. Scene c) show that the predicted annotation is larger than the actual ship size for small ships.

Predicted ships using DRBox

a) Larger ships

b) Small ship



Figure 9.2: Visualization of DRBox results in two scenes. Scene a) consists of two larger ships. The adjacent dock interferes with the predictions. Scene b) consist of one smaller ship that is successfully predicted.

Predicted ships in harbor area

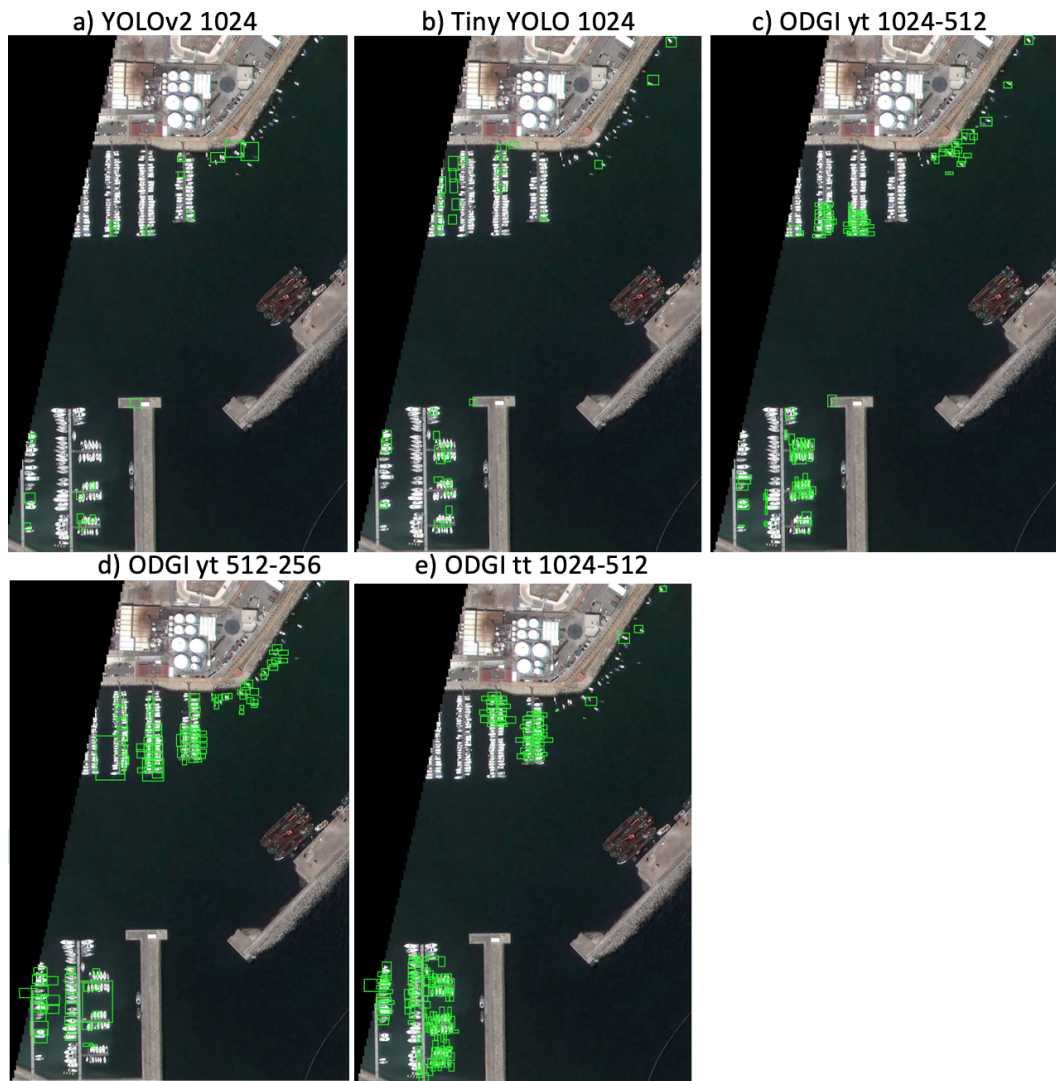


Figure 9.3: Visualization of one example tile projecting a harbor area with corresponding predictions generated using the reference models YOLOv2 1024, Tiny YOLO 1024, ODGI yt 1024-512, ODGI yt 512-256 and ODGI tt 1024-512. Image dimensions are manipulated with respect to the original tile for visualization, as part of the tile is outside the satellite product.

Predictions in area with high ship density

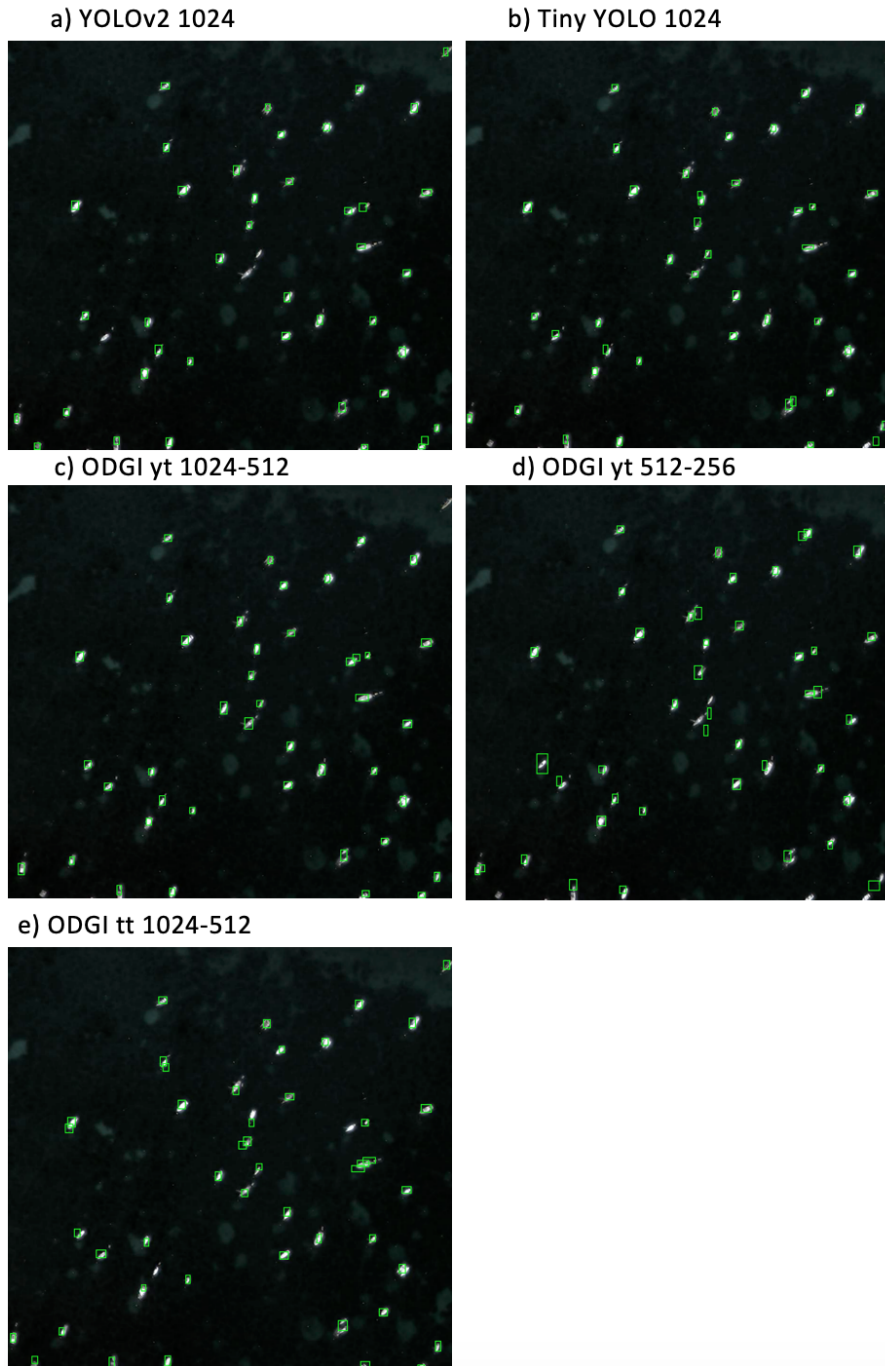


Figure 9.4: Visualization of one example tile projecting an area of high ship density, and corresponding predictions generated using the reference models YOLOv2 1024, Tiny YOLO 1024, ODGI yt 1024-512, ODGI yt 512-256 and ODGI tt 1024-512. Image dimensions are manipulated with respect to the original tile for visualization.

/10

Experiments on Novel Methods

This chapter will present experiments on the novel models presented in chapter 8. As introduced in chapter 8, both Oriented YOLOv2 and Oriented Tiny YOLO can be used with different grid cell sizes. The same applies for the OODGI pipeline. Different complexities of these three models are experimented on in this chapter, and discussed and analyzed further in part V. Experimental setups will first be introduced in chapter 10.1, before experimental results are presented in section 10.2.

10.1 Experimental Setup

Similarly as for experiments on reference models, the hyperparameters τ_{IoU} and τ_c are included in all models. Although the IoU is calculated somewhat different as for reference models experiments, $\tau_{IoU} = 0.5$ is used for an easier comparison to other models. All novel models are grid-based detectors¹, and all experiments are conducted using the exact SuperView data set, therefore, $\tau_c = 0.4$ is ascertained here already.

1. Although OODGI is a hybrid of grid- and proposal-based architecture, grid-based detectors constitutes the individual stages.

Based on unreported experiments and suggested values for λ_{coord} and λ_{noobj} in the standard YOLOv2 model, $\lambda_{orientation} = 1$ is a reasonable value, causing the optimization process to converge. This is used for all experiments in this chapter. Similarly as for reference models experiments, only positive image tiles are used in the training data set, i.e. images that have at least one annotated object.

The implementations of Oriented YOLOv2, Oriented Tiny YOLO and OODGI are extensions of the ODGI code base², written in the Tensorflow framework (Abadi et al., 2015), but is unfortunately not published due to commercial interests.

10.1.1 Oriented YOLOv2

The same experimentation principles as described in section 9.1.5 for the standard YOLOv2 applies for experiments on Oriented YOLOv2. Identical regularization strategies, data augmentation (with different implementation), weight initialization, and activation functions are used in these experiments as well. The difference is the new loss function, expressed in equation 8.1, different IoU concept and other modifications that are described in chapter 8. A simple linear activation function is applied to the introduced θ_ϕ parameter, as no mapping of this regressed parameter is desired.

Hyperparameters are determined similarly as for the YOLOv2 experiments, in addition to $\lambda_{orientation} = 1$. Experiments are performed on Oriented YOLOv2 1024 and Oriented YOLOv2 512. These models are selected because they have the highest potential of producing adequate and interesting results.

10.1.2 Oriented tiny-YOLO

Experiments on Oriented Tiny YOLO are conducted identically as for Oriented YOLOv2, but now using a more light-weight CNN as described in section 8.3. The same grid cell sizes as described in section 9.1.3 apply. Unlike the reference model experiments, Oriented Tiny YOLO 512 is now included because of the promising results of Oriented Tiny YOLO 1024 and Oriented YOLOv2 512 in Table 10.1.

10.1.3 OODGI

A similar experimental setup as for ODGI, described in section 9.1, applies for the OODGI experiments. When OODGI was presented, it was argued that $S = 2$ stages in the pipeline is sufficient for the normal remote sensing data set. Because the SuperView is considered a normal remote sensing data set, $S = 2$ is used in the OODGI experiments. The modifications described in chapter 8 are applied. The new loss function, expressed in equation 8.3, is used for

2. <https://github.com/ameroyer/ODGI>.

optimization. Except for $\lambda_{orientation} = 1$, all hyperparameters are identical as for ODGI.

For a holistic perspective on the novel ODGI model, all ODGI yt 1024-512, ODGI yt 512-256, ODGI tt 1024-512 and ODGI tt 512-256 are included in the experiments. These models are considered the most relevant compositions with the highest potential for producing adequate results. Unfortunately, there are some indications that the ODGI implementation has some errors and is not fully completed. These indications included large losses for stage two, ϕ_2 , and failing results.

10.2 Experimental Results

In this section, experimental results for the setups described in the previous section is presented. Inference time, AP and the number of parameters (network weights) for the different setups are presented in Table 10.1.

Oriented Tiny YOLO 512 and ODGI tt 512-256 are not included in the visual prediction examples. Recall that basic ships, larger ships, areas of high ship density and harbor scenes are considered scenarios of special interest. These mentioned scenarios are together considered to constitute a holistic perspective of potential scenarios. Figure 10.1 visualizes a basic scene with corresponding predictions by the novel models. Figure 10.2 exemplifies predictions by the novel models in a large ship scenario. Figure 10.3 visualizes one example scene where all models proved to be struggling, and with corresponding predictions. Figure 10.4 displays one example harbor, known as a difficult scene, and corresponding predictions. To generate Figure 10.4, a lower confidence score threshold τ_c and IoU threshold τ_{IoU} were used to achieve a more explanatory figure.

Particularly noteworthy details from Table 10.1 are the favorable results of Oriented YOLOv2 512, the weak performance of Oriented YOLOv2 1024, and the weak performance of all ODGI models. The experimental results on reference models in Table 9.1 report precisions that are relatively expected from the model complexities. Some of the reported precisions in Table 10.1 are somewhat more unexpected. A noteworthy detail from Figure 10.1 and 10.2 is that ODGI yt 512-256 is the only model missing objects in both these scenes. In Figure 10.3, most models miss ships and predict inaccurate ship orientations. All models predict poor annotations for the harbor scene in Figure 10.4, but the ODGI models barely manages to predict any objects at all.

Table 10.1: Results of experiments with the SuperView data set described in chapter 7 on the novel object detection models presented in chapter 8. Inference times are obtained with a GeForce RTX 2080 Ti GPU.

Model	Inference time [s]	AP	# parameters
Oriented YOLOv2 1024	0.157	0.400	51 M
Oriented YOLOv2 512	0.0635	0.451	51 M
Oriented Tiny YOLO 1024	0.175	0.4286	11 M
Oriented Tiny YOLO 512	0.054	0.417	11 M
OODGI yt 1024-512	0.171	0.375	62 M
OODGI yt 512-256	0.0634	0.351	62 M
OODGI tt 1024-512	0.165	0.341	22 M
OODGI tt 512-256	0.0558	0.244	22 M

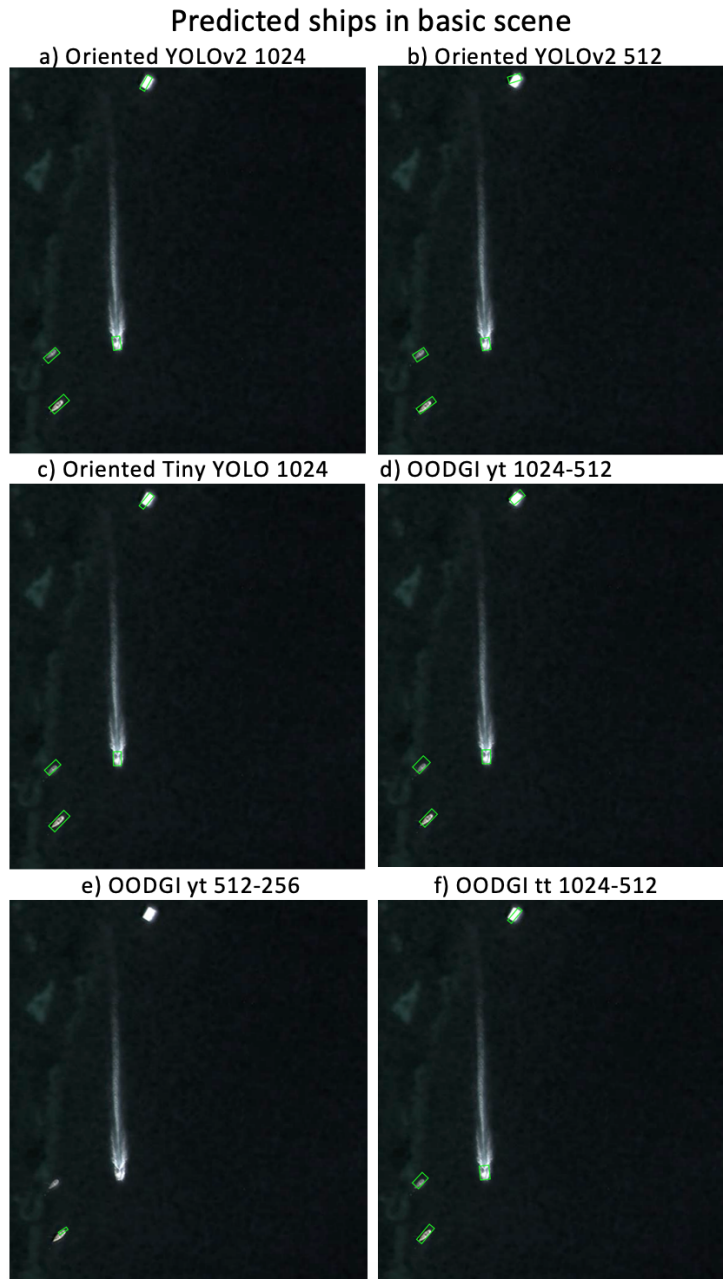


Figure 10.1: Visualization of one example basic scene with corresponding predictions by the novel models. Predictions are generated using Oriented YOLOv2 1024, Oriented YOLOv2 512, Oriented Tiny YOLO 1024, OODGI yt 1024-512, OODGI yt 512-256 and OODGI tt 1024-512. Image dimensions are manipulated with respect to the original tile for visualization.

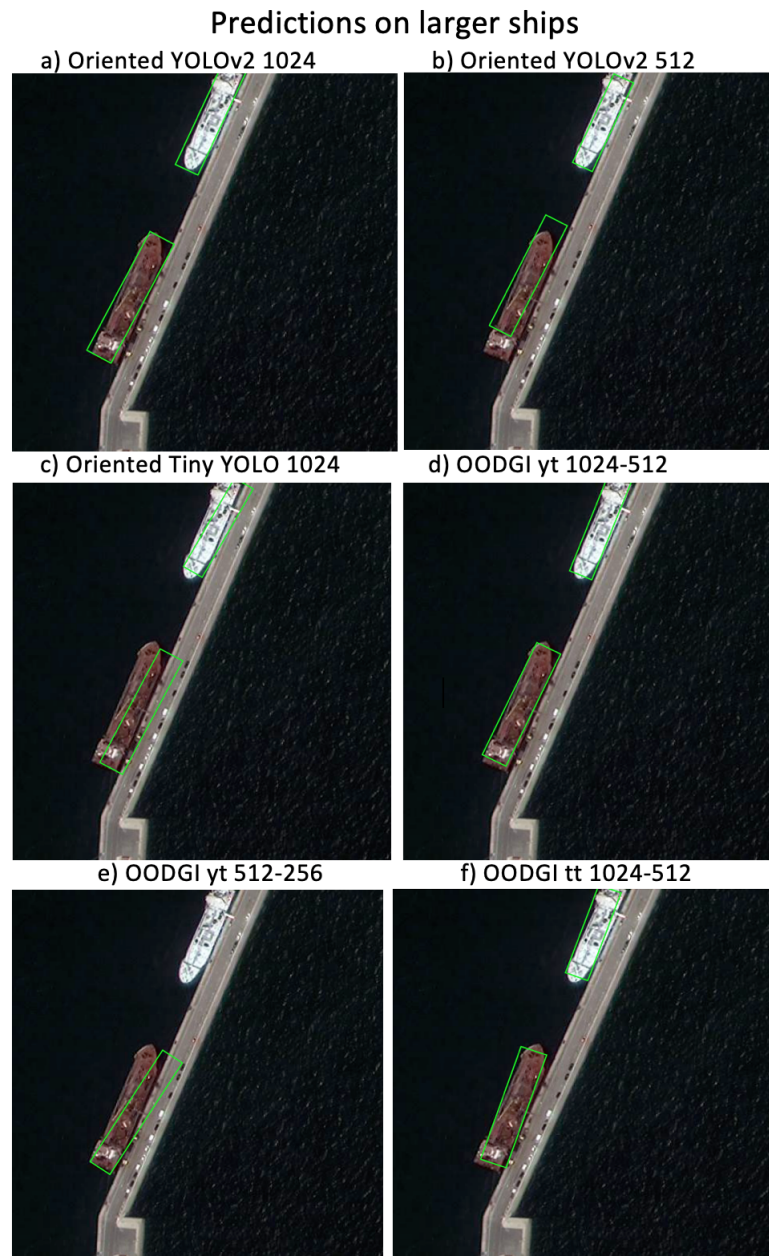


Figure 10.2: Visualization of one scene displaying large ships and corresponding predictions by the novel models. Image dimensions are manipulated with respect to the original tile for visualization.

Predictions in challenging scene

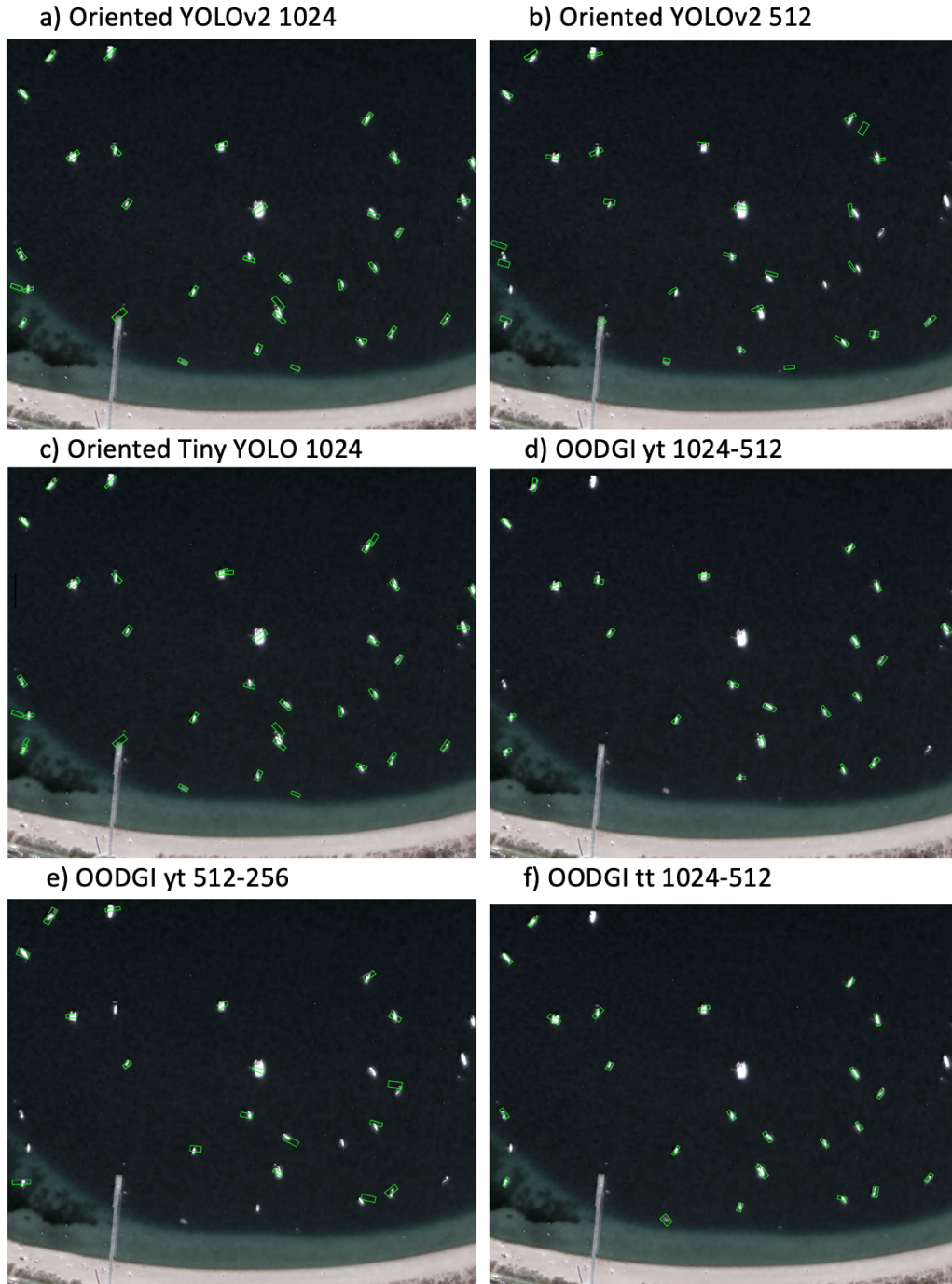
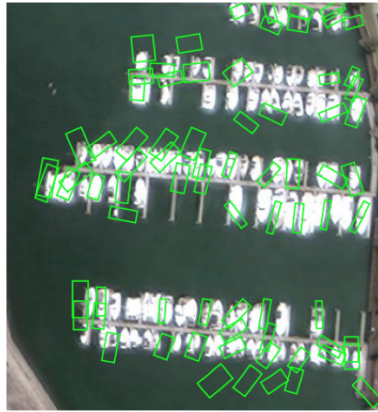


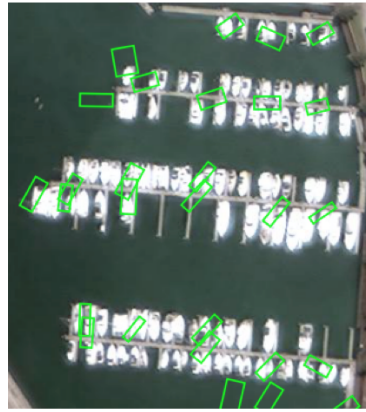
Figure 10.3: Visualization of one example scene that all novel models find troublesome. Image dimensions are manipulated with respect to the original tile for visualization.

Predictions in harbor scene

a) Oriented YOLOv2 1024



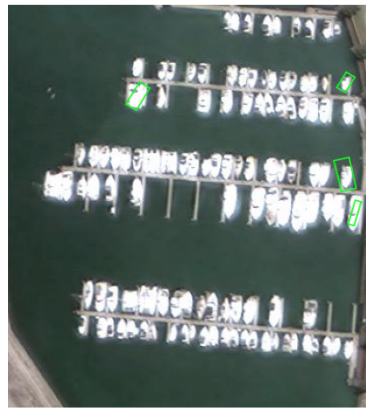
b) Oriented YOLOv2 512



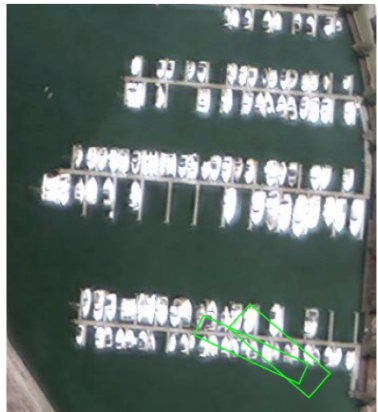
c) Oriented Tiny YOLO 1024



d) OODGI yt 1024-512



e) OODGI yt 512-256



f) OODGI tt 1024-512



Figure 10.4: Visualization of one example harbor scene with corresponding predictions by the novel models. Image dimensions are manipulated with respect to the original tile for visualization.

/ 11

Additional Experiments on Main Models

This chapter will present some interesting analyses on the main models that are not covered by the previous two chapters. Section 11.1 will present results that are specifically interesting within the field of ship detection, and will encompass both reference and novel models. Section 11.2 studies the stochasticity of validation results, inspired by several somewhat unforeseen precision results in Table 10.1.

11.1 Error Analysis on Main Models

It was stated in section 4.7.1 that false alarms, errors, and correct predictions are of specific interest in the scientific field of ship detection. These error measures are often crucial when choosing a ship detection model. This section studies these three error measures on the main reference and novel models. The main models are selected according to promising results in Table 9.1 and 10.1, with diversity in mind, and are the models included for visualization in section 9.2, Figure 9.4 and 9.3, and section 10.2.

A prediction is considered to overlap a ground truth ship if the intersection between these boxes is greater than 10^{-4} . This will include all overlapping boxes (with overlap greater than 10^{-4}), and is not considering the size of the boxes. In theory, a prediction covering the entire image will coincide with all

ground truth ships. Fortunately, no such predictions have emerged in analyzed images. All hyperparameters and thresholds are set according to described values in section 9.1 and 10.1.

True positive rate (tp-rate) is equivalent to correct prediction rate. False positive rate (fp-rate) is equivalent to false alarm rate (also equivalent to Type I errors). False negative rate (fn-rate) is equivalent to the rate of misses (also equivalent to Type II errors). The ideal situation is tp-rate = 1.0, fp-rate = 0.0 and fn-rate = 0.0. The results of this error analysis are given in Table 11.1, and graphed in Figure 11.1. Noteworthy details are the improved rates for Oriented YOLOv2 and Oriented Tiny YOLO compared to the non-oriented equivalents, and the failing fp-rates for the OODGI models.

Table 11.1: Performance measures of main reference and novel models. The included measures are true positive rate (tp-rate), false positive rate (fp-rate) and false negative rate (fn-rate).

Model	tp-rate	fp-rate	fn-rate
YOLOv2 1024	0.743	0.046	0.104
Tiny YOLO 1024	0.756	0.086	0.163
ODGI yt 1024-512	0.875	0.046	0.126
ODGI yt 512-256	0.761	0.081	0.148
ODGI tt 1024-512	0.890	0.110	0.092
Oriented YOLOv2 1024	0.861	0.139	0.074
Oriented YOLOv2 512	0.885	0.115	0.094
Oriented Tiny YOLO 1024	0.857	0.143	0.170
OODGI yt 1024-512	0.593	0.007	0.120
OODGI yt 512-256	0.388	0.113	0.255
OODGI tt 1024-512	0.593	0.007	0.076

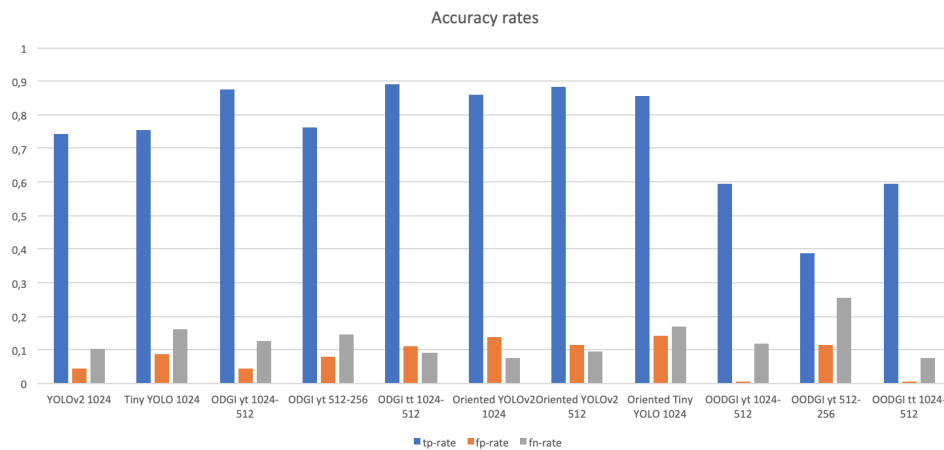


Figure 11.1: Chart of performance measures. Visualization of the values in Table 11.1.

11.2 Stochasticity Analysis

Inspired by several somewhat unforeseen precision results in Table 10.1, and the previously mentioned model stochasticity introduced by batch processing (stated in section 4.5.1), an analysis of varying results in different training sessions seemed appropriate. This section studies the AP results when evaluating 9 different training sessions. It is performed on Oriented YOLOv2 1024, as this model may be considered one of the main presented novelties. In addition, this model converges faster than the OODGI variants.

The exact same split of training, validation and test data is used for all 9 experiments, and they are trained with an equal amount of epochs, same batch size, and identical hyperparameters.

The results are summarized in Figure 11.2. The sample mean of these 9 AP results is $\bar{x} = 0.401$, and the sample standard deviation is $\hat{\sigma} = 0.024$. Worth noting is that the sample mean of these data points (0.401) is relatively similar to the reported AP result of Oriented YOLOv2 in Table 10.1 (0.400). The results will be discussed further in chapter 12.

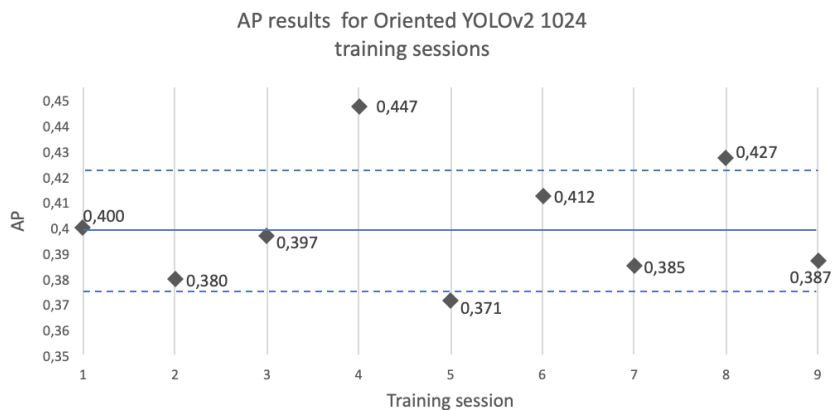


Figure 11.2: AP results from 9 different Oriented YOLOv2 1024 training sessions. Sample mean $\bar{x} = 0.401$ (solid line) and sample standard deviations (dotted lines) are added. The sample standard deviation is $\hat{\sigma} = 0.024$.

Part V

Discussion and Conclusion

/12

Discussion

The results presented in part IV will be discussed in this chapter. An important and interesting issue when studying model performance is the emphasis of speed versus accuracy. Slower models are typically more precise, and vice versa, often making this issue decisive when choosing a model. The issue of speed versus accuracy trade-off will be reviewed in the beginning.

12.1 Speed versus Accuracy Trade-Off

Model inference speed can often have an impact on model selection. Processing time per image ranges typically from tens of milliseconds to almost a second, dependent on image resolution and model complexity (Huang et al., 2017). Long inference time tends to produce more precise results, and vice versa. Several proposal-based object detectors are initially slower but more accurate, but may achieve increased inference speed if the maximum number of region proposals are restricted. For the ship detection problem, where harbor scenes similar to the one visualized in Figure 1.1 are recurring, this is not feasible.

Ship detection service providers are typically interested in running near real-time analyses. Recall from section 3.4 that a SuperView satellite product consists of $\sim 3.2 \times 10^4$ pixels. When using a tile size of 1024 pixels, one satellite product is equivalent to $\sim 10^3$ images tiles. With an inference time of one second, the total computation time will demand over 16 minutes. A paradigm says that “coarse predictions are better than none” (Royer and Lampert, 2020). Consequently, a simpler non-complex model may often be considered adequate

for given problems. For ship detection, such a model will typically result in a higher false alarm and miss rate.

12.2 Discussion of Results

Chapter 9 and 10 presented results on reference and novel models, respectively, and chapter 11 presented error and stochasticity results on selected models. The following sections will discuss these results, the overall model performances, and a general discussion.

12.2.1 Reference Models

The studied reference models include Faster R-CNN, DRBox, YOLOv2 1024, YOLOv2 512, Tiny YOLOv2 1024, ODGI yt 1024-512, ODGI yt 512-256 and ODGI tt 1024-512. Table 9.1 presented main experimental results on these reference models. Faster R-CNN and YOLOv2 512 achieves the highest and lowest AP results, respectively. YOLOv2 512 and DRBox have the lowest and highest inference time, respectively. Although being a grid-based detector, DRBox operates with an inference time as high as 2.364 seconds. This is probably largely because of the computer-intensive IoU calculation. Because this calculation is also performed during inference and testing, it is considered to be too slow to be used in most real world applications. Liu et al. (2017a) presents a significantly lower inference time than reported here, which indicates that a slow implementation is used in the current work. Although Royer and Lampert (2020) claims that the motivation behind the ODGI model is to reduce computational intensity, and hence reducing inference time, while sustaining satisfying precision, Table 9.1 reveals increased precision and competitive speed as compared to the regular YOLOv2 and Tiny YOLO models. The table also indicates that models which was initialized using pretrained weights (Faster R-CNN and DRBox) tend to result in a higher precision.

Figure 9.1 displays Faster R-CNN predictions at three typical scenes. In Subfigure (a), the two adjacent ships are so close that the NMS step silences one true prediction because of overlap. This exemplifies the drawback of using BBox to describe arbitrarily oriented objects. Subfigure (b) visualizes a failing attempt to predict ships in a harbor. Subfigure (c) shows that the predicted bounding box is not small and precise enough to correctly enclose the small ship. This is a recurring situation for small ship predictions, and is due to limitations of the Faster R-CNN, specifically the receptive field. Figure 9.2 visualizes sufficient results for two larger adjacent ships, and one smaller ship, predicted by the DRBox.

Figure 9.3 visualizes YOLOv2, Tiny YOLO and ODGI predictions in a demanding harbor scene. YOLOv2 1024 and Tiny YOLO 1024 (Subfigure (a) and (b), respectively) performs poorly, whereas the ODGI models performs slightly

better. Recall from section 9.1 that $\tau_1^{test} = 8$ crops are used during inference for these experiments. This is clearly evident in Subfigure (c), (d) and (e). Only a limited number of regions in the image (8 regions) are refined at stage two ϕ_2 . This is a typical scene with many object groupings and a large τ_1^{test} would have been appropriate. There is no indications of $S = 2$ stages being to shallow for the scenarios encountered in this SuperView data set. The number of misses is unacceptable in all these results. Figure 9.4 visualizes a scene with numerous smaller ships in a confined area, and corresponding predictions by selected reference models. YOLOv2 1024 and Tiny YOLO 1024 (Subfigure (a) and (b), respectively) generate reasonably good predictions, but with some misses and misplacements. Most ships are not sufficiently close to be considered as groupings by the ODGI models. From the figure, it appears that ODGI yt 1024-512 produced the most solid predictions for this scene.

12.2.2 Novel Models

The variants of novel models used for experimenting are Oriented YOLOv2 1024, Oriented YOLOv2 512, Oriented Tiny YOLO 1024, Oriented Tiny YOLO 512, OODGI yt 1024-512, OODGI yt 512-256, OODGI tt 1024-512 and OODGI tt 512-256. Table 10.1 presents the main experimental results on these novelties. Oriented YOLOv2 512 and OODGI tt 512-256 achieve the highest and lowest AP scores, respectively. Oriented Tiny YOLO 512 and OODGI yt 1024-512 operate with the lowest and highest inference time, respectively. An unanticipated result is the robust precision of Oriented YOLOv2 512 and Oriented Tiny YOLO 512 as compared to Oriented YOLOv2 1024 and Oriented Tiny YOLO 1024, respectively. These versions operate with half the number of prior boxes, and have a significantly shorter inference time. As a result of an implementation fault that could not be resolved within the time frame of the project, all OODGI models achieve disappointing precisions.

A comparison of the Oriented YOLOv2 and Oriented Tiny YOLO in Table 10.1 with the non-oriented equivalents in Table 9.1 reveals fairly comparable results. The more lightweight reference model YOLOv2 512 achieved significant improved precision when extended to Oriented YOLOv2 512.

Figure 10.1 visualizes one example basic scene with multiple smaller ships, and corresponding predictions by selected novelties. Oriented YOLOv2 512 (Subfigure (b)) reports one misoriented prediction, and OODGI yt 512-256 (Subfigure (e)) has three misses. Except for this, the results are relatively robust. Figure 10.2 illustrates predictions in a scene with two larger ships and an adjacent dock. Again, OODGI yt 512-256 (Subfigure (e)) is the only model missing a ship. Other than this, the predictions are sufficient.

The scene considered in Figure 10.3 is found troublesome by all models used for experimenting. Few of these predictions are considered to have successfully

recognized orientations. Oriented YOLOv2 512, OODGI yt 512-256 and OODGI tt 1024-512 (Subfigure (b), (e) and (f), respectively) have some misses. Beyond this, most objects are detected, but with inaccurate rotations. The reason is assumed to be objects being too small and cluttered. Most of these ships are not sufficiently close to be considered object groupings by the OODGI models.

Figure 10.4 visualizes one example of a troublesome harbor scene and corresponding predictions by novel models. All Oriented YOLOv2 and Oriented Tiny YOLO versions (Subfigure (a), (b) and (c)) produce numerous false alarms, have several misses and inaccurate shapes and rotations. The OODGI models have an unsatisfactory amount of misses and produce incomplete results. This is a typical scene with many intermediate predictions that need refinement in stage two ϕ_2 . Because of the assumed implementation fault, these refined predictions become incorrect, and hence achieve a low confidence score which causes them to be silenced during NMS. The implementation fault has been attempted to be corrected, but could not be completed due to time limitations. Because of large losses in the stage two object detection network ϕ_2 during training, and incorrect predictions especially in areas of high object density, the implementation fault is assumed to be at the stage transition or when formatting predictions from the concluding stage, ϕ_2 . It can be debated whether the imprecision is caused by weakness in the introduced model, rather than being an implementation fault. However, the very large losses in stage two ϕ_2 compared to stage one ϕ_1 indicate that this may not be the cause.

Recall from section 8 that angle prediction in all novel models is performed using a direct regression approach. It was also stated that this approach was suggested with some uncertainty about how well it was going to perform, and that the experiments in chapter 10.2 should reveal its performance. The results in Table 10.1 and the predictions visualized in section 10.2 describes overall adequate predictions of object rotations, indicating that solving this task with direct regression is sufficient. Based on satisfactory results for most scenes in section 10.2, which indicates a potential of sufficient object orientation recognition, it is tempting to think that the weaker results in Figure 10.3 could be improved with more training data.

Recall from the DRBox introduction in section 6.3.2.1 and the IoU calculation for novel models, described in chapter 8, that the novel models maps RBoxes to BBoxes when calculating IoU, whereas DRBox uses an approximate approach for determining this measure. Although DRBox operates with a significant higher inference time, it appears to produce a higher precision (ref. Table 9.1). Thus, testing this approximated IoU calculation for the novel models would have been an interesting experiment.

In section 5.2.5, it was stated that the RBox omits background noise which is included in the BBox. This is claimed to result in better decision-making, and is the origin of Hypothesis 1: *Annotating objects in remote sensing images using rotatable bounding boxes gives technical improvements leading to increased precision over traditional bounding boxes for deep learning models, and gives a visually more prominent description*. An AP comparison of architectures using RBoxes and architectures using BBoxes in Table 9.1 and 10.1 may indicate a slightly improved AP for oriented architectures. However, this is not statistically tested. Numerous figures have exemplified the visual and descriptive advantages of adopting RBoxes.

As a conclusion of this section, Oriented YOLOv2 and Oriented Tiny YOLO have shown competitive and promising results. They report relatively good precisions, while operating relatively fast during inference, and appealing visualizations have resulted from these models. OODGI reports weaker results for precision. It operates adequately in scenes with lower object density, but struggles once the density increases.

12.2.3 Additional Analysis

The error and stochasticity analysis presented in chapter 11 will be reviewed and discussed in this section. The error analysis includes selected reference and novel models. The stochasticity analysis is performed on the Oriented YOLOv2 1024 model.

12.2.3.1 Error Analysis

Table 11.1 and Figure 11.1 reveals that OODGI tt 1024-512 has the highest rate of correct predictions, OODGI yt 1024-512 and OODGI tt 1024-512 share the medal for lowest false alarm rate, and Oriented YOLOv2 1024 has the lowest number of misses.

Figure 11.1 also reveals the gratifying indicator that the Oriented YOLOv2 and Oriented Tiny YOLO variants have a higher rate of correct predictions compared to the non-oriented equivalents. A hypothesis test to evaluate whether this is a statistically valid statement would have been highly interesting, but is omitted as multiple subexperiments would have been needed, and due to time limitations. In return, these oriented models do also appear to have a higher rate of false alarms and misses. The OODGI models operates in general with a low rate of correct predictions, few false alarms and a high rate of misses.

12.2.3.2 Stochasticity Analysis

Figure 11.2 shows a plot of the AP evaluation results for 9 individually trained Oriented YOLOv2 1024 models, used to study the variance between training sessions. All models are trained identically with the exact same data. The only factor causing variations in the AP results are, to the authors best knowledge, random batch compositions and successions from the stochastic weight initialization. The sample mean of these 9 AP results is $\bar{x} = 0.401$, and the sample standard deviation is $\hat{\sigma} = 0.024$. This sample mean is relatively similar to the reported AP result of Oriented YOLOv2 in Table 10.1 (0.400). However, the sample standard deviation $\hat{\sigma} = 0.024$ suggests that this result is sampled from a stochastic variable, and thus that this similarity is a coincidence.

Achieving these varying results when using the exact same data may indicate that the optimization process had not yet converged. The validation results were alternating for numerous epochs before the training session was completed, and there was no indications of further stabilization.

These experimental results may indicate that the data set is not of sufficient scale. One troublesome sample in the test data set can make a big impact on the evaluation process. A greater extent of the data set would possibly have decreased the observed variance.

12.3 General

A statistical analysis of a null hypothesis concerning similarity of AP in the novel oriented versions compared to the standard non-oriented equivalents would have been highly interesting. Specifically, this could have been used to conclude on Hypothesis 1 concerning technical improvements in the model when adopting RBoxes. Because the oriented models are not resulting from the same stochastic process, multiple subexperiments are necessary for doing such a statistical analysis. The same applies for the non-oriented models. This is omitted due to time limitations.

In part II, there was some mentions of the topic of ship detection using segmentation networks. The inclusion of a segmentation-based model in the experiments could have been valuable for the sake of comparison. However, as discussed in section 7.2, there is a class imbalance on pixel level. When applying segmentation-based ship detection models, it is common to ignore land areas using a *land mask*. This will, i.a., guide the network to get a monotone understanding of the two classes of interest: ship and ocean (background). Experimentation on segmentation-based architectures are omitted due to time limitations, as it is outside the scope of this thesis.

The advantage of the ability to describe shape and rotation of oriented objects is, in this thesis, presented as highly desirable. The number of problems where this is applicable may still be debatable. The advantages of using RBoxes over traditional BBoxes must be taken into consideration when choosing an object detector. Predicted rotations have in some examples in this thesis appeared to be somewhat unstable, which is assumed to be a particular problem if the training data set extent is not sufficient. Besides this, the benefits of using RBoxes have proven to be many, and is the motivation behind most experiments in this thesis. Scenes of high object density have also been particularly studied in this thesis. Most such scenes have proven to be difficult to handle for all reviewed architectures. The necessity of detecting ships at harbors or near shore may be debatable.

Optical remote sensing images, and the accompanying high resolution and fine details, have worked well for ship detection experiments. The resolution has generally been sufficient for rotation recognition. After different appearances of the SuperView data was studied in section 7.1, it became clear that lots of training data is vital when training deep neural networks on this data. Varying experimental results may indicate that better precisions and accuracy rates could be obtained with more training data.

None of the models studied in this thesis predict a ship heading $\in [0, 360)^\circ$, i.e., they do not differentiate bow and stern of a ship. This is partly because DRBox is not doing so, and comparability is desired. All concepts introduced for the three neural network architectures supports such an expansion. The annotations in the SuperView data set also supports this extension¹.

1. The orientations were mapped to $[0, 180)^\circ$ as a pre-processing step to be used in the studied models.

/ 13

Concluding Remarks

The thesis has studied ship detection using deep learning in optical remote sensing images. Five reference models of different complexities have been studied. Some of these have again been decomposed and implemented with different complexities for experimentation. Three novel neural network architectures predicting object rotations have been proposed: Oriented YOLOv2, Oriented Tiny YOLO and Oriented Object Detection with Grouped Instances (OODGI). The first two demonstrated adequate results, whereas OODGI needs some implementation adjustments to demonstrate its full potential. Different compositions of these architectures have been reviewed and are used for experimentation. A complete SuperView data set suited for deep learning applications has been washed and finalized.

Hypothesis 1 claimed that adopting RBoxes gives technical improvements leading to increased precision and gives a more prominent description over the traditional BBoxes. The novel models Oriented YOLOv2 and Oriented Tiny YOLO have given indications of improved precision and rate of correct predictions compared to the non-oriented equivalents, and hence a technical improvement. This indication originates from a comparison of APs in Table 9.1 and 10.1, and accuracy rates in Figure 11.1. However, this is not statistically tested, and therefore cannot be used to conclude Hypothesis 1. Numerous figures have exemplified the visual and descriptive advantages of adopting RBoxes, and hence concludes parts of Hypothesis 1.

Hypothesis 2 claimed that the advantage of grouped objects being more visually salient than individual objects will result in increased precision when applying deep learning architectures based on object groupings to the SuperView data set. YOLOv2 and Tiny YOLO had a low inference time, and acceptable results. However, ODGI reported increased performance compared to the baseline models YOLOv2 and Tiny YOLO, while operating with equivalent inference time. These results are presented in Table 9.1. Thus, ODGI achieved improved precision, concluding Hypothesis 2.

Faster R-CNN reported the overall best precision results. However, this model is relatively slow and struggles with enclosing smaller ships. DRBox reported a high inference time, but fairly good precision.

Hypothesis 3 claimed that expanding a deep learning architecture based on object groupings to predict RBoxes (over the traditional BBoxes) will describe objects more orderly, while safeguarding the advantages of utilizing object groupings. The expansion of an object detector based on grouped object instances to detection of orientation has theoretically been achieved, as a full description of OODGI highlighting the necessary modifications of ODGI was presented. However, the failing experimental results indicates that the advantages of utilizing object groupings are not safeguarded, and the investigation of Hypothesis 3 cannot be considered as concluded.

All models reviewed in this thesis are struggling with harbor scenes, and scenes of high object density. This will remain as an unsolved issue¹. The performance consistency of Oriented YOLOv2 1024 using the exact same SuperView data set has been reviewed. Varying results was achieved over 9 different training sessions used in the experiment. It was discussed that the reason for this varying results were random batch compositions and successions from the stochastic weight initialization. It was also discussed that a greater training data set would possibly have decreased the variance.

1. It was predicted already in introduction chapter 1 that no object detectors will be able to process such challenging scenes perfectly.

13.1 Future Work

Numerous interesting aspects have emerged in this thesis. A further study, an extension of this thesis, would have been pleasant. Some moments were omitted due to time limitations, and some moments were excluded because of irrelevance. Ideas for future works include:

- Rectification of the OODGI implementation fault.
- Initialization of the novel models with pretrained weights to obtain improved performance.
- Experimentation on using the IoU approximation approach used in DR-Box for our novel models.
- Inclusion of 360° orientation recognition.
- Inclusion of vessel type classification.
- Expansion of the SuperView data set.
- Experimentation on Faster R-CNN with all classification components excluded to obtain improved detection performance.
- Performance comparison between bounding box-based and segmentation-based detectors on the SuperView data set.
- Implementation of a complete pipeline that splits the satellite image into convenient image tiles, processes these image tiles individually, and reassembles the results. This process must not lose decisive information.

This concludes the thesis ✓

/14

Appendix

14.1 RBox to BBox Mapping

Originally, annotations in the SuperView data set are given in the $\theta_{RBox} = \{\theta_x, \theta_y, \theta_w, \theta_l, \theta_\phi\}$ (xywh-format) format. Several models expect the $\theta_{BBox} = \{\theta_x, \theta_y, \theta_w, \theta_h\}$ format. A common representation is also the $\theta_{BBox} = \{\theta_{x_min}, \theta_{y_min}, \theta_{x_max}, \theta_{y_max}\}$ format. A mapping from the RBox to a BBox representation is therefore necessary. This mapping is also performed several times for some of the models studied in this thesis. The following Python3 function is designed to perform this mapping:

```
import numpy as np
def convert_RBox_to_BBox(RBox, xywh_format=True):
    x, y, w, l, r = RBox
    hyp = np.sqrt((w / 2)**2 + (l / 2)**2)

    x1 = int(x - hyp * np.cos(r) - (w / 2) * np.sin(r))
    y1 = int(y - hyp * np.sin(r) - (w / 2) * np.cos(r))

    x2 = int(x + hyp * np.cos(r) - (w / 2) * np.sin(r))
    y2 = int(y + hyp * np.sin(r) + (w / 2) * np.cos(r))

    x3 = int(x + hyp * np.cos(r) + (w / 2) * np.sin(r))
    y3 = int(y + hyp * np.sin(r) - (w / 2) * np.cos(r))

    x4 = int(x - hyp * np.cos(r) + (w / 2) * np.sin(r))
```

```

y4 = int(y - hyp * np.sin(r) + (w / 2) * np.cos(r))

x_min = np.min([x1, x2, x3, x4])
x_max = np.max([x1, x2, x3, x4])

y_min = np.min([y1, y2, y3, y4])
y_max = np.max([y1, y2, y3, y4])

if xywh_format:
    return (x_max + x_min)/2, (y_max + y_min)/2,
           x_max - x_min, y_max - y_min
else:
    return x_min, y_min, x_max, y_max

```

14.2 Crop-Relative Annotation Mapping

The following Python3 code snippet is designed to map one annotation from original coordinate system to a crop relative coordinate system. Notations are described in equation 8.2.

```

import tensorflow as tf
rbox_sub = tf.concat([crop_mins_x,
                     crop_mins_y,
                     tf.zeros_like(crop_w),
                     tf.zeros_like(crop_h),
                     tf.zeros_like(crop_r)], axis=-1)

rboxes -= rbox_sub
new_rbox_w = (w_rbox*(tf.sin(rbox_angle)
                    + tf.cos(rbox_angle)))
            /tf.maximum(1e-6, crop_width)
new_rbox_l = (l_rbox*(tf.cos(rbox_angle)
                    + tf.sin(rbox_angle)))
            /tf.maximum(1e-6, crop_height)
rboxes = tf.concat([rboxes[..., 0], # rbox_x
                  rboxes[..., 1], # rbox_y
                  new_rbox_w,
                  new_rbox_l,
                  rbox_angle], axis=-1)

```

Bibliography

- Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.
- Alpaydin, E. (2014). *Introduction to machine learning (3rd Edition)*. MIT press.
- Bahrampour, S., Ramakrishnan, N., Schott, L., and Shah, M. (2015). Comparative study of deep learning software frameworks. *arXiv preprint arXiv:1511.06435*.
- Balduzzi, M., Pasta, A., and Wilhoit, K. (2014). A security evaluation of ais automated identification system. In *Proceedings of the 30th annual computer security applications conference*, pages 436–445. ACM.
- Bamler, R. and Hartl, P. (1998). Synthetic aperture radar interferometry. *Inverse problems*, 14(4):R1.
- Brenn, T., Gjøvik, L.-P., Rasmussen, G., Bauna, T., Kampffmeyer, M., Jenssen, R., and Anfinson, S. (2019). Operationalizing ship detection using deep learning. <https://www.cmre.nato.int/msaw-2019-home/msaw2019-papers/1398-msaw2019-brenn-operationalizingshipdetectionusingdeeplearning/file>.
- Campbell, J. B. and Wynne, R. H. (2011). *Introduction to Remote Sensing (5th Edition)*. The Guilford Press, A Division of Guilford Publications, Inc., 72 Spring Street, New York, NY 10012.
- Chen, X., Xiang, S., Liu, C.-L., and Pan, C.-H. (2014). Vehicle detection in satellite images by hybrid deep convolutional neural networks. *IEEE Geoscience and remote sensing letters*, 11(10):1797–1801.

- Cheng, G. and Han, J. (2016). A survey on object detection in optical remote sensing images. *ISPRS Journal of Photogrammetry and Remote Sensing*, 117:11–28.
- Cheng, G., Zhou, P., and Han, J. (2016). Learning rotation-invariant convolutional neural networks for object detection in vhr optical remote sensing images. *IEEE Transactions on Geoscience and Remote Sensing*, 54(12):7405–7415.
- Chollet, F. et al. (2015). Keras. <https://keras.io>.
- Curry, H. B. (1944). The method of steepest descent for non-linear minimization problems. *Quarterly of Applied Mathematics*, 2(3):258–261.
- Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 886–893. IEEE.
- Dragosevic, M. V. and Vachon, P. W. (2008). Estimation of ship radial speed by adaptive processing of radarsat-1 fine mode data. *IEEE Geoscience and Remote Sensing Letters*, 5(4):678–682.
- Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(Jul):2121–2159.
- Eldhuset, K. (1996). An automatic ship and ship wake detection system for spaceborne sar images in coastal regions. *IEEE transactions on Geoscience and Remote Sensing*, 34(4):1010–1019.
- Everingham, M., Van Gool, L., Williams, C. K., Winn, J., and Zisserman, A. (2010). The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338.
- Girshick, R. (2015). Fast R-CNN. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448.
- Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587.
- Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international*

- conference on artificial intelligence and statistics*, pages 249–256.
- Gonzalez, R. C. and Woods, R. E. (2018). *Digital Image Processing (4th Edition)*. Pearson Education Limited, Edinburgh Gate, Harlow, Essex CM20 2JE, England.
- Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning (1st Edition)*. MIT Press, Cambridge, MA.
- Grabner, H., Nguyen, T. T., Gruber, B., and Bischof, H. (2008). On-line boosting-based car detection from aerial images. *ISPRS Journal of Photogrammetry and Remote Sensing*, 63(3):382–396.
- Han, J., Zhang, D., Cheng, G., Guo, L., and Ren, J. (2014). Object detection in optical remote sensing images based on weakly supervised learning and high-level feature learning. *IEEE Transactions on Geoscience and Remote Sensing*, 53(6):3325–3337.
- Hansen, M. A. (2019). Affinity-guided image-to-image translation for unsupervised heterogeneous change detection. Master’s thesis, UiT Norges arktiske universitet.
- He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.
- He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.
- Hinton, G., Srivastava, N., and Swersky, K. (2012a). Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, 14(8).
- Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012b). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.
- Howard, J. and Gugger, S. (2020). fastai: A layered api for deep learning. *Information*, 11(2):108.
- Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., Fischer, I., Wojna, Z., Song, Y., Guadarrama, S., et al. (2017). Speed/accuracy trade-offs for modern convolutional object detectors. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7310–7311.

- Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.
- Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. (2014). Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*.
- Jiang, Y., Zhu, X., Wang, X., Yang, S., Li, W., Wang, H., Fu, P., and Luo, Z. (2017). R2cnn: rotational region cnn for orientation robust scene text detection. *arXiv preprint arXiv:1706.09579*.
- Kaggle (2018). *Dataset For Airbus Ship Detection Challenge*. <https://www.kaggle.com/c/airbus-ship-detection>.
- Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
- Kong, T., Yao, A., Chen, Y., and Sun, F. (2016). Hypernet: Towards accurate region proposal generation and joint object detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Leal-Taixé, L. and Roth, S. (2019). *Computer Vision – ECCV 2018 Workshops: Munich, Germany, September 8-14, 2018, Proceedings*. Number poeng 3 in Lecture Notes in Computer Science. Springer International Publishing.
- LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436–444.
- Li, F.-F. and Perona, P. (2005). A Bayesian hierarchical model for learning natural scene categories. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 524–531. IEEE.
- Lin, H., Shi, Z., and Zou, Z. (2017a). Fully convolutional network with task partitioning for inshore ship detection in optical remote sensing images. *IEEE Geoscience and Remote Sensing Letters*, 14(10):1665–1669.
- Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B., and Belongie, S. (2017b). Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125.
- Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P. (2017c). Focal loss for dense object detection. In *Proceedings of the IEEE international conference on*

- computer vision*, pages 2980–2988.
- Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014). Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer.
- Liu, G., Sun, X., Fu, K., and Wang, H. (2013). Interactive geospatial object extraction in high resolution remote sensing images using shape-based global minimization active contour model. *Pattern Recognition Letters*, 34(10):1186–1195.
- Liu, L., Ouyang, W., Wang, X., Fieguth, P., Chen, J., Liu, X., and Pietikäinen, M. (2018). Deep learning for generic object detection: A survey. *International Journal of Computer Vision*, pages 1–58.
- Liu, L., Pan, Z., and Lei, B. (2017a). Learning a rotation invariant detector with rotatable bounding box. *arXiv preprint arXiv:1711.09405*.
- Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C. (2016a). Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer.
- Liu, Y., Cui, H.-Y., Kuang, Z., and Li, G.-Q. (2017b). Ship detection and classification on optical remote sensing images using deep learning. In *ITM Web of Conferences*, volume 12, page 05012. EDP Sciences.
- Liu, Z., Wang, H., Weng, L., and Yang, Y. (2016b). Ship rotated bounding box space for ship extraction from high-resolution optical satellite images with complex backgrounds. *IEEE Geoscience and Remote Sensing Letters*, 13(8):1074–1078.
- Long, J., Shelhamer, E., and Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440.
- Longbotham, N., Pacifici, F., Malitz, S., Baugh, W., and Camps-Valls, G. (2015). Measuring the spatial and spectral performance of worldview-3. In *Hyperspectral Imaging and Sounding of the Environment*, pages HW3B–2. Optical Society of America.
- Makantasis, K., Karantzalos, K., Doulamis, A., and Doulamis, N. (2015). Deep supervised learning for hyperspectral data classification through convolutional neural networks. In *2015 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, pages 4959–4962. IEEE.

- Masi, G., Cozzolino, D., Verdoliva, L., and Scarpa, G. (2016). Pansharpening by convolutional neural networks. *Remote Sensing*, 8(7):594.
- Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814.
- Nesterov, Y. E. (1983). A method for solving the convex programming problem with convergence rate $o(1/k^2)$. In *Dokl. akad. nauk Sssr*, volume 269, pages 543–547.
- Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch.
- Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.
- Qian, N. (1999). On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151.
- Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Redmon, J. and Farhadi, A. (2017). Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271.
- Redmon, J. and Farhadi, A. (2018). Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*.
- Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99.
- Robbins, H. and Monro, S. (1951). A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407.

- Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer.
- Rottmann, K. (2003). *Matematisk Formelsamling*. Spektrum forlag.
- Royer, A. and Lampert, C. (2020). Localizing grouped instances for efficient detection in low-resource scenarios. In *The IEEE Winter Conference on Applications of Computer Vision*, pages 1727–1736.
- Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088):533–536.
- Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. (2015). Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252.
- Sai, W., Ren, J., and Jidong, Z. (2019). Superview-1-china’s first commercial remote sensing satellite constellation with a high resolution of 0.5 m. *中航天 (英文版)*, 19(1):31–38.
- Sandland, A. (2019). An explorative analysis of ais data and optical satellite data aimed at identifying an automated training data collection approach for ship detection using deep learning.
- Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520.
- Santurkar, S., Tsipras, D., Ilyas, A., and Madry, A. (2018). How does batch normalization help optimization? In *Advances in Neural Information Processing Systems*, pages 2483–2493.
- Shi, S., Wang, Q., Xu, P., and Chu, X. (2016). Benchmarking state-of-the-art deep learning software tools. In *2016 7th International Conference on Cloud Computing and Big Data (CCBD)*, pages 99–104. IEEE.
- Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.
- Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2013). Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*.

- Tao, D., Doulgeris, A. P., and Brekke, C. (2016). A segmentation-based cfar detection algorithm using truncated statistics. *IEEE Transactions on Geoscience and Remote Sensing*, 54(5):2887–2898.
- Theodoridis, S. and Koutroumbas, K. (2009). *Pattern Recognition (4th Edition)*. Academic Press, Elsevier, 525 B Street, Suite 1900, San Diego, California, 92101-4495, USA.
- Uijlings, J. R., Van De Sande, K. E., Gevers, T., and Smeulders, A. W. (2013). Selective search for object recognition. *International journal of computer vision*, 104(2):154–171.
- Vivone, G. (2014). Multispectral and hyperspectral pansharpening.
- Vivone, G., Alparone, L., Chanussot, J., Dalla Mura, M., Garzelli, A., Licciardi, G. A., Restaino, R., and Wald, L. (2014). A critical comparison among pansharpening algorithms. *IEEE Transactions on Geoscience and Remote Sensing*, 53(5):2565–2586.
- Wen, W., Wu, C., Wang, Y., Chen, Y., and Li, H. (2016). Learning structured sparsity in deep neural networks. In *Advances in neural information processing systems*, pages 2074–2082.
- Xia, G.-S., Bai, X., Ding, J., Zhu, Z., Belongie, S., Luo, J., Datcu, M., Pelillo, M., and Zhang, L. (2018). Dots: A large-scale dataset for object detection in aerial images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3974–3983.
- Xu, S., Fang, T., Li, D., and Wang, S. (2009). Object classification of aerial images with bag-of-visual words. *IEEE Geoscience and Remote Sensing Letters*, 7(2):366–370.
- Yang, X., Sun, H., Fu, K., Yang, J., Sun, X., Yan, M., and Guo, Z. (2018). Automatic ship detection in remote sensing images from google earth of complex scenes based on multiscale rotation dense feature pyramid networks. *Remote Sensing*, 10(1):132.
- Yang, X., Yang, J., Yan, J., Zhang, Y., Zhang, T., Guo, Z., Sun, X., and Fu, K. (2019). Scrdet: Towards more robust detection for small, cluttered and rotated objects. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 8232–8241.
- Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

- Zhang, D., Han, J., Cheng, G., Liu, Z., Bu, S., and Guo, L. (2014). Weakly supervised learning for target detection in remote sensing images. *IEEE Geoscience and Remote Sensing Letters*, 12(4):701–705.
- Zhang, Z., Guo, W., Zhu, S., and Yu, W. (2018). Toward arbitrary-oriented ship detection with rotated region proposal and discrimination networks. *IEEE Geoscience and Remote Sensing Letters*, 15(11):1745–1749.
- Zhong, P. and Wang, R. (2007). A multiple conditional random fields ensemble model for urban area detection in remote sensing optical images. *IEEE Transactions on Geoscience and Remote Sensing*, 45(12):3978–3988.