Faculty of Sciences

Department of mathematics

Ship detection using SAR imagery and machine learning techniques

A thesis submitted in fulfillment of the requirements for the Master's degree in mathematics

Academic year 2022–2023

Author:
Delphine Remacle

Supervisor:
Murielle Kirkove

Co-Supervisors:
Arnout Van Messem
Alexander Barth

Acknowledgements

First of all, I would like to express my deepest gratitude to my supervisor, Murielle Kirkove, for giving me the opportunity to work on this interesting subject and for her constant support and availability throughout this work. I would also like to express my sincere thanks to Arnout Van Messem, who agreed to supervise this master's thesis, without him it would not have been possible. I would like to extend my gratitude to Alexander Barth who also agreed to take part in this project. I would like to thank them all for their interest in this work, their judicious advice and their comments whenever I needed them.

This endeavour would not have been possible without the *Liège Space Center* who gave me the opportunity to work on this subject and who provided me with the necessary material to complete it, I would like to thank them for this.

I would like to extend my thanks to all the members of the jury for agreeing to read this document and to take part in the defence of this thesis.

I am also grateful to Quentin Glaude for his review and valuable advice on SAR imagery.

Then, I wanted to thanks Robert Sip for his availability and his sound advice on informatics, especially on Linux, which was completely new to me.

My thanks also go to my parents, who gave me the opportunity to pursue these studies, and my boyfriend Josly for their unfailing support throughout these years of study.

Last but not least, I would like to thank my friend Marie for her precious help throughout our studies and for her presence during our endless study sessions.

Contents

Introduction

This master's thesis is part of a project carried out by several students, which goal is to find a suitable program of ship detection from Synthetic-Aperture Radar (SAR) imagery by using machine learning techniques. The first student has determined a dataset composed of SAR images suitable for ship detection, see [61]. In particular, she assessed its pertinence for this task by testing machine learning methods, RetinaNet and Faster R-CNN. [12]

In the theoretical part I, we will give an in-depth introduction to SAR imagery and machine learning, but now, we provide a brief explanation of these concepts.

Ship detection is essential for maritime surveillance purposes. Many reasons justify the interest in this field, a major one being to monitor the traffic of ships to avoid collisions between them. Another reason is to detect a ship if a maritime rescue is required. It can also be useful in port management or for monitoring cargo transport. Moreover, illegal activities such as fishing in prohibited areas, immigration or human trafficking could be controlled.

There exists an official vessel tracking system, the automatic identification system (AIS), but it does not work in case of illegal activities. It is therefore interesting to be able to locate ships engaged in such activities or which simply have a defective AIS system. To do this, **SAR imagery** can be used, as it provides high-resolution images of the Earth taken from satellites in orbit. For instance, two SAR images are illustrated in Figure 1, in which the bright elements are ships. In this case, the ships are easily identifiable, but it is not always that clear. The first chapter, see 1, provides more details about SAR imagery and ship detection using it.

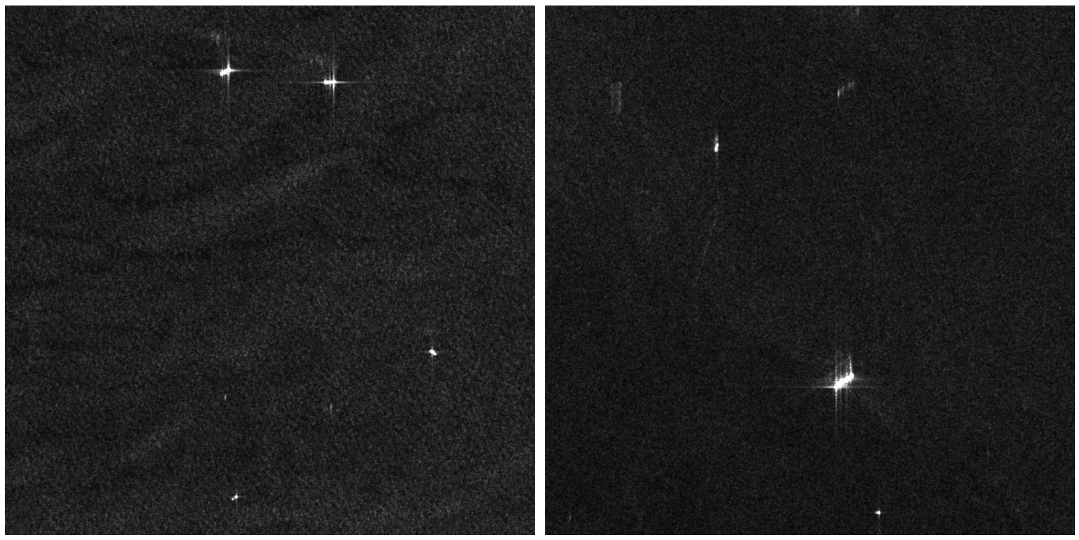


Figure 1: Images 01_9_23.jpg and 01_14_13.jpg of the LS-SSDD-v1.0 dataset. © [61]

The second chapter of this theoretical part, see 2, concerns a specific form of **artificial intelligence**. This branch of computer science aims to create machines that can simulate human intelligence. In this master's thesis, the focus will be on **machine learning**, which is itself a branch of artificial intelligence that does not use explicit programming to solve a certain task. Indeed, machine learning is based on mathematical and statistical concepts allowing the computer to learn from observations. The idea is that, based on these data, the learning algorithm will try to improve its accuracy to realise a particular task without it being explicitly programmed. In other words, a learning algorithm is a method, corresponding to a set of instructions, that enables a computer program to mimic how a human being improves at a specific task by learning from examples.

In this dissertation, the focus is on two related branches of artificial intelligence, namely deep learning and computer vision. **Deep learning** is a branch of machine learning in which the initial objective was for algorithms to mimic the behaviour of the human brain instead of the general behaviour of a human being, and this through artificial neural networks. The term "deep" indicates that more complex architectures than in traditional machine learning are used.

Computer vision is a branch of artificial intelligence that focuses on visual data (numerical images and videos). This master's thesis aims to find an appropriate algorithm to detect ships on images. This task is called **object detection**. For this purpose, deep learning algorithms can be used. Some techniques of computer vision will then be introduced in the following.

In Part II, the practical results are discussed. In particular, three deep learning techniques were tested: YOLO, versions 3 and 4, and Mask R-CNN. In the paper [61], the authors compared a number of object detection algorithms, including Faster R-CNN, which was one of the best. According to the theory, Mask R-CNN, which is an improvement on Faster R-CNN, performs better, that is why testing this method is interesting. According to [61], YOLO version 3 did not provide good performance measures. However, the YOLO family of models is known for its ability to provide state-of-the-art performance for real-time object detection. Interest was therefore focused on this method to determine whether better results could be obtained. We also tested the version 4 of YOLO, not mentioned in [61], as it is expected to perform better than version 3.

The model optimisation of these three methods is given in Chapter 4. Comparisons between them are presented in Chapter 5. The codes written during this master's thesis are explained in Chapter 6. Finally, the conclusion and the limitations of our study are given in Chapters 8 and 9.

Part I

Theoretical bases

Chapter 1

Tools for maritime surveillance purposes

The purpose of this chapter, which is based on the references [12], [33] and [56], is to explain existing tools for ship detection for maritime surveillance purposes. Section 1.1 introduces the automatic identification system (AIS), which is an official vessel tracking system that does not work in case of illegal activities. Then, Section 1.2 explains Synthetic-Aperture Radar (SAR) imagery, from which some illegal activities can be detected.

1.1 AIS

Automatic identification system (AIS) is a system for automatically locate and identify vessels, mainly to monitor them and to avoid collisions. Normally, vessels transporting passengers and those over 300 gross tons must be equipped with this system. The AIS transponders on these ships send information to the AIS receptors placed on other vessels, on shores and in harbour areas. These receptors detect the transponders up to about 40 km off shore. At a range greater than 40 km off shore, the standard, i.e. terrestrial, AIS system no longer works and another method is therefore needed to detect vessels in open seas, far from the shore.

In this case, the **Space-based AIS (S-AIS)** is rather used, which has a very large detection range, around 3000 km in diameter. The satellites detect the AIS signals and then send them to some ground stations, where post-processing is performed.

Thus, cooperative vessels can be detected using terrestrial and space-based AIS. However, AIS remains useless in the case of non-cooperative vessels, i.e. if they deliberately do not embark transponders on board or send AIS signals in order not to be detected. For instance, this is the case of ships used for human trafficking. In this situation, another solution is required.

1.2 SAR imagery

This section presents **spaceborne SAR imagery**, where SAR stands for Synthetic-Aperture Radar, which is a type of radar that allows the creation of high resolution images containing information captured about the ground. A brief discussion follows on the use of SAR imagery for ship detection.

SAR sensor

Spaceborne remote sensing is the branch that focuses on Earth observation using satellites placed in orbit around it. Not all satellites placed in orbit are designed for Earth observation. If so, they are called **Earth-Observation (EO)** satellites and can, for example, monitor the atmosphere, the weather or the Earth's surface. For the latter, the corresponding satellites are equipped with a **sensor** that aims to capture information from the ground such as oceans, seas, ships, land surfaces or towns. A sensor performs **remote sensing** by remotely obtaining information about the Earth's surface. More precisely, there is no physical contact with the ground because electromagnetic pulses are sent, interact with the matter and then, the amount of energy that is reflected back, or "backscattered", is recorded. There are two types of sensors, **passive** and **active**. The difference lies in the source of the signal they use to observe the object. Passive sensors rely on the reflection of solar radiation while active sensors use their own energy source. The latter have the advantage of operating at any time, day or night.

Radar sensors are active and use wavelengths at centimetre to metre scale, enabling them to observe through clouds and the atmosphere. Some categories of wavelength, separated as bands, are more widely used than others in SAR imagery. These are X-band, C-band, L-band and P-band. [41] In particular, what interests us in the following is the **SAR sensor** that operates in C-band. This band is a part of the microwave range in which the central frequency is 5.4Ghz (which corresponds to wavelength of 5.5cm), and for which the Earth's atmosphere is transparent.

Unlike other radars, **Synthetic-Aperture Radar** provides high spatial resolution images. The spatial resolution measures the fineness of the smallest details present in an image. The higher the resolution, the finer the detail. As an image is composed of pixels, it is said, in the case of satellite images, that each pixel represents a certain area of the Earth's surface. In the case of radars, to obtain finer resolution for a given wavelength, the antenna must be larger, which limits the spatial resolution of the radar. In the case of SAR, the concept of synthetic-aperture is used, the idea being to take advantage of the motion of the antenna due to the velocity of the satellite to take successive measurements of the same region but from different positions, simulating a much larger antenna. [41]

There are many missions using SAR sensors whose sensor characteristics differ. In particular, one of these is on board of Sentinel-1, which interests us and is described in the next section. In this case, the SAR sensor operates in C-band and has the following advantages: it can operate at any time of the day (day/night observation), the Earth's atmosphere is transparent to it, it covers large areas with medium to high resolution, it is an operational ship monitoring system working in near real time and it is capable of monitoring ships in estuaries and harbour areas.

Sentinel-1

Sentinel-1 is one of the constellations of the Copernicus project, which is an Earth observation program of the European Union developed by the European Space Agency (ESA) to continuously monitor the Earth on the following themes: atmosphere, marine, land, climate change, security and emergency. Sentinel-1 can be used for all these topics, but in particular it allows land and ocean monitoring. In fact, it consists of two polar-orbiting satellites, Sentinel-1A and Sentinel-1B, which orbit the Earth at the same altitude of 693 km, see Figure 1.1. The orbit has a 12 day repeat cycle (it takes 12 days to return to its initial orbit), but when the two satellites, placed in the same orbit with a 180 degree phase shift, are used, the repeat cycle is divided in two and is therefore of 6 days. [16] This means that when both satellites are used, Sentinel-1 offers a six-day coverage of the Earth, allowing environmental monitoring in almost real-time. This revisit time can be reduced by using overlapping regions from different tracks. However, as Sentinel-1B is no longer in service since December 2021, the information mentioned above only concerns the period from 2016 to 2021.

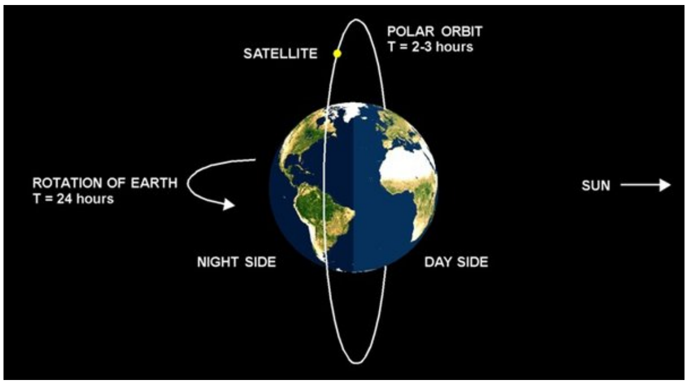


Figure 1.1: Representation of a polar-orbiting satellite. © [10]

Other advantages of using Sentinel-1 include the availability of the collected data on a public platform within a maximum of 24 hours, free of charge. Depending on the user's needs, different levels of pre-processed data are available. These include **Ground Range Detected (GRD)** data. These are in Level-1 format, which follows Level-0 corresponding to the raw unfocused data collected by the satellite in binary format. At this level of pre-processing, the phase, information about the distance between the sensor and a target, is lost, but the amplitude, the quantity of the transmitted signal that returns to the sensor, is preserved. In contrast, Single Look Complex (SLC) data preserve the phase and amplitude of the signal. SLC data, which are also in Level-1 format, lead to images in which each pixel is represented by a complex number, unlike GRD data which use a real number. [15] In addition, the pixels in a GRD image are square, which is not necessarily the case with SLC. The dataset used in the following is made up of GRD data, which is appropriate because the phase of the signal is not necessary for ship detection. Finally, software for managing the different levels of pre-processed data from Sentinel-1 is made available to the public.

Signal polarisation

The **polarisation of a signal** refers to the direction of electromagnetic waves. The direction of polarisation is the orientation of the plane in which the waves propagate. In general, SAR sensors generate horizontally (H) or vertically (V) polarised waves. This kind of radar enables the selection of the polarisation method used when transmitting and receiving the signal. The following table summarises the existing methods for transmitted and received signals in horizontal or vertical polarisation.

		Signal reception	
		Horizontal	*Vertical*
Signal transmission	*Horizontal*	HH	HV
	Vertical	VH	VV

For Sentinel-1, the main mode of data acquisition in maritime surveillance areas is the **interferometric wide swath (IW)** mode, which is suitable for observing large areas. When GRD is used, three different spatial resolutions are available (Full, High and Medium), but the IW mode restricts this choice to High (HR) and Medium (MR) resolutions. So, for instance, a pixel could correspond to an area of 10 metres by 10 metres for a HR or 40 metres by 40 metres for a MR. The images of the dataset used in the following consist of GRD data collected in IW mode. Furthermore, this mode allows the use of HH, HV, VV and VH polarised waves although not all at the same time. Thanks to this combination of signal acquisitions methods, IW mode has the particularity of covering a ground width (swath) of 250 km. Furthermore, the polarisation method used and the structure of the object's surface influence the amount of backscattered energy.

Ship detection with SAR imagery

This part first introduces the ship signature in SAR images. It then presents the interest in using deep learning techniques for ship detection and why it makes sense to use these techniques to detect ships from SAR images.

As briefly mentioned at the end of last section, the surfaces observed by the sensor have different structures, which implies different properties of the backscattered signal. In the case of scattering on a rough surface, such as the ocean, which is relatively flat despite some imperfections due to waves, a small amount of the energy reflected by the surface returns to the satellite sensor and the surface appears black. On the other hand, a large amount of the energy is reflected towards the sensors when it hits a surface producing a double-bounce effect. This means that the signal is backscattered twice, creating a strong backscattering reflexion towards the sensors. The information on such surfaces therefore appears through bright pixels. This applies to surfaces with low rugosity such as to towns, land or ships, for example. When some regions can not be reached by the signal, they appear black since there is no backscattered signal collecting some information to describe them. Figure 1.2 illustrates this. Black pixels correspond to the ocean. White pixels in the ocean correspond to ships. Finally, the different shades of grey in the bottom left corner correspond to a piece of land.

Initially, ship detection in SAR images was performed using traditional algorithms that require human intervention. Many algorithms were available, including constant false alarm ratio (CFAR)-based.[1] [36] As these methods are based on SAR images in which most ships can be distinguished from the background, they are potentially capable of detecting non-cooperative ships, i.e. those not using the AIS system. [33]

Nowadays, advances in deep learning techniques in various fields and mainly the release of SAR Ship Detection Dataset (SSDD) have raised interest in the application of such methods to ship detection using SAR images. [36] [62] Moreover, as mentioned in the article [61], these techniques almost dominate traditional methods because they are more accurate, faster and require less human intervention. Therefore, using deep learning methods would lead to the automation of the detection process while achieving great performance.

In SAR images, as ships correspond to bright pixels on the ocean that appears black,

[1]For further information, the article [36] outlines the development of SAR ship detection from such traditional algorithms to deep learning algorithms.

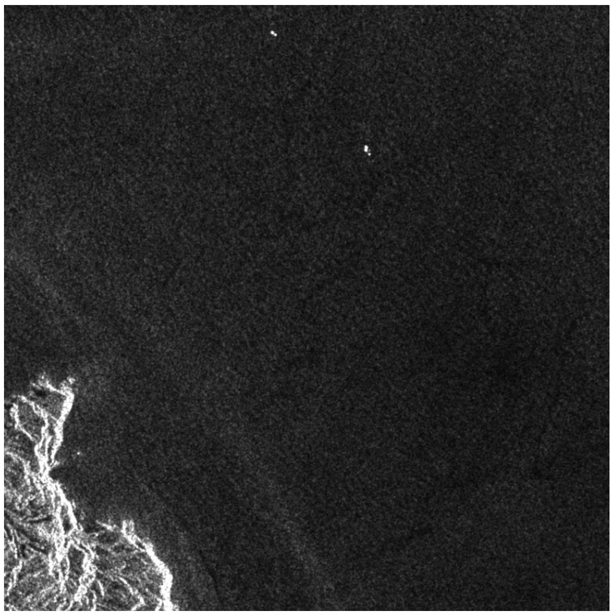


Figure 1.2: Image 01_10_24.jpg from LS-SSDD-v1.0 dataset. Black pixels correspond to the ocean. White pixels in the ocean correspond to ships. Finally, the different shades of grey in the bottom left corner correspond to a piece of land. © [61]

the idea would be for the deep learning algorithm to learn how to recognise those bright pixels that correspond to ships. It therefore makes sense to use such images for ship detection. However, the deep learning algorithm will encounter several difficulties, such as distinguishing a ship with turbines, debris or any other element appearing as bright pixels, since they all appear as bright pixels against a black background, or distinguishing ships near the coast or in a harbour, since both ships and land appear as bright pixels.

Chapter 2

Machine learning

This chapter provides the theoretical background on machine learning. First, Section 2.1 introduces its principle. Section 2.2 presents the theory behind deep learning. After that, the principle of computer vision and object detection is explained in Section 2.3. Finally, Section 2.4 presents some machine learning methods, including those that will be tested in the practical part.

2.1 Bases of machine learning

This section first introduces the basic principle of machine learning. Then, it presents the linear regression algorithm and the method of least squares on which it is based. Next, the mathematical concepts used in machine learning are briefly discussed. Finally, the development of this field is outlined.

2.1.1 Machine learning principle

As explained previously, **machine learning**[1] is a branch of artificial intelligence in which programs aim to mimic the way a human being improves at a specific task. For this purpose, the learning algorithm is given data and its aim is to improve its ability to perform the task based on that data. It is therefore important to select an appropriate dataset for the task. For example, a dataset made up of images of cats and dogs is suitable if the goal is to classify such images. The choice of the learning algorithm is also important. There are many different algorithms, such as linear regression which will be presented in detail in next section, support vector machine (SVM) which can be used for classification, k-nearest neighbours (kNN) or random forest, which can be used for classification or regression.

When one chooses a learning algorithm, a model linked to it is selected. For instance, an algorithm based on a linear regression initialises a mathematical equation of a linear regression with unknown parameters. This example is described in more detail in next section.

Hence, the goal of the algorithm is to adjust the values of the model parameters, which are typically randomly initialised, so that the resulting model describes the data as well as possible. To determine how well the model fits the data, it is necessary to measure the error rate between the outputs predicted by the model and the observations. This measure is computed by the **loss function**. The objective of the algorithm is therefore

[1]This section was written using the information available in [21] and [27].

to find the values of the model parameters that minimise the error rate on the available data. To achieve this, it will modify the parameter values until it obtains those that lead to the lowest error rate on the data. It can therefore be said that the algorithm learns from the data, and this learning period is called **training**. Actually, this step corresponds to a principle of learning by experience. To get a more concrete idea, one can compare it to the period when a baby learns how to walk: at the beginning, he falls all the time and he does not know how to walk, but as he gains experience, he learns how it works and finally, he is able to walk.

Finally, if the amount of data provided to the algorithm is larger, learning will be improved, making more accurate predictions, which means that it will perform better. Indeed, the model will be more fitting if there are more data available.

Remark 2.1.1. The difference between a classical algorithm and a learning algorithm is that the first one is not able to improve itself by determining which are the best parameter values during training.

Once the learning step is complete, the optimised model can be used in practice for the same task as in the learning step. New data can be fed to the trained algorithm, which will use the optimised model to produce new predictions for that data. This corresponds to the **test** step, during which the quality of the model's predictions can be assessed.

Example 2.1.1. For instance, suppose we have data consisting of emails with a label indicating whether it is a spam or not and provide these data to a learning algorithm. In this way, it is as if we explicitly tell the machine that such an email corresponds to a spam and such an email corresponds to a ham message. Once the algorithm is trained and has a model, a new email without label "spam" or "ham" can be provided to it and it will be able to predict if the email is spam or not. Indeed, during training, the algorithm will have learned which are the characteristics of a spam message and which are those of a ham. The model provided by the algorithm is then able to catch these characteristics. Hence, given a new email, it will search if it finds features of spam or ham and based on that, it will classify the email. This example of spam detection will be used several times throughout this work as an illustration.

A task such as spam messages detection can be referred to as **supervised learning**: the information initially available includes labelled data, i.e. the answer to the task for that data is available. The labels can be discrete or continuous respectively, in which case it is a task of **classification** or **regression** respectively. In a classification task, the model produces a discrete output, which refers to a category, whereas in a regression task, the model produces a continuous output. In the case of spam detection, the labels are either "spam" or "not spam" and this corresponds to a classification task.[2] The linear regression presented in the next section is an example of a regression task.

Some tasks do not have labelled data, in this case we speak of **unsupervised learning**. Thus, we are only interested in the data, without looking at the responses associated with it, which was the case in supervised learning. The objective is to determine the structure present in the data. There are for example the clustering or dimension reduction techniques such as principal component analysis, which are also techniques used in multivariate statistics. Back to the example, it is as if the "spam" and "not spam" labels were not available. In this case, the algorithm should find a structure in the data and classify them in two

[2]However, there is no guarantee that a class will emerge during the test step of a new input.

clusters corresponding to the spam messages on one side and to the ham messages in the other side.

Nowadays, a machine learning model can perform other practical tasks such as translating text or speech, recognising an object in an image, being an automatic assistant like Siri[3] or managing the driving of an autonomous car. For example, Netflix also uses such a tool when it suggests a series that a user might like based on what they have already watched and based on the same profile of other users.

2.1.2 Least squares example

In this section based on [26] and [27], a classical algorithm in machine learning, the **(Ordinary) Least Squares Linear Regression**, is considered. It will also serve as an illustration of the minimisation of the loss function, i.e. of the prediction error rate of the model.

This algorithm of linear regression uses the least squares method. A linear regression aims to predict an output value given several input variables. Even if it is quite simple, this method is still used in practice because it allows to analyse trends in data and to do basic predictions. This method works quite well in the case of small datasets or large datasets not too complex. In addition, several improvements of this technique exist.

Least squares principle

Before analysing how the least squares method is used in machine learning, this section recalls what the current least squares technique is. For this purpose, we investigate the ordinary case of this method. The goal is to estimate the unknown parameters of a linear regression model, i.e. of a linear relationship between explained and explanatory variables. Least squares principle aims to minimise the sum of squared deviations between values predicted by the linear model and the explained variables.

Given a dataset $\{\mathbf{x}_i, y_i\}_{i=1}^n$ composed of n observations where, for $i \in \{1, ..., n\}$, the column vector $\mathbf{x}_i$ contains m elements, i.e. $\mathbf{x}_i = (x_{i1}, x_{i2}, ..., x_{im})^T$, and the response y_i is a scalar. The notations

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}_1 & \mathbf{x}_2 & ... & \mathbf{x}_n \end{pmatrix}^T = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1m} \\ x_{21} & x_{22} & \cdots & x_{2m} \\ \vdots & & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nm} \end{pmatrix}$$

for the matrix of n vectors $\mathbf{x}_i$ and $\mathbf{y} = (y_1, ..., y_n)^T$ for the column vector of the n responses y_i are used. The general linear regression model is defined as

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon},$$

where the linear approximation of each y_i is given by

$$\boldsymbol{\beta}^T \mathbf{x}_i = \beta_1 x_{i1} + ... + \beta_m x_{im}$$

with $\boldsymbol{\beta} = (\beta_1, ..., \beta_m)^T$ a column vector of m unknown elements to determine and where $\boldsymbol{\epsilon} = (\epsilon_1, ..., \epsilon_n)^T$ is a column vector of n independent error terms such that $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$.

[3]Siri is an intelligent assistant available on Apple products that understands the user's verbal instructions and responds and/or performs the user's request.

If $n > m$, the system is over-determined, i.e. it has more equations than unknown variables, and, in general, there is no exact solution to such a problem. So, the goal is to determine the coefficients $\boldsymbol{\beta}$ of $\mathbf{X}\boldsymbol{\beta}$, which allow to estimate as best as possible the vector $\mathbf{y}$. This is where the least squares method comes in. Indeed, the minimisation of the squared deviations between the vector $\mathbf{y}$ and the approximation $\mathbf{X}\boldsymbol{\beta}$ of this vector must be computed, which corresponds to minimise $||\mathbf{y} - \mathbf{X}\boldsymbol{\beta}||^2$.

Reformulation of the problem 2.1.1. *In the same conditions as above of an over-determined system, with $\boldsymbol{X}^T\boldsymbol{X}$ a squared matrix assumed to be of maximal rank, in order to approximate as best as possible the vector $\mathbf{y}$ by the linear relation $\boldsymbol{X\beta}$, the least squares principle suggests to minimise the sum of squared deviations between the vector $\mathbf{y}$ and the approximation $\boldsymbol{X\beta}$ of this vector, so to minimise $||\mathbf{y} - \boldsymbol{X\beta}||^2$.*

This is equivalent to determine $\widehat{\boldsymbol{\beta}}$ such that

$$\widehat{\boldsymbol{\beta}} = \arg\min_{\beta} ||\mathbf{y} - \boldsymbol{X\beta}||^2$$

and the optimal vector $\widehat{\boldsymbol{\beta}}$ is then given by

$$\widehat{\boldsymbol{\beta}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\mathbf{y}.$$

Proof. The term to minimise can be rewritten as follows

$$\begin{aligned} f(\boldsymbol{\beta}) = ||\mathbf{y} - \mathbf{X}\boldsymbol{\beta}||^2 &= (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) \\ &= (\mathbf{y}^T - \boldsymbol{\beta}^T\mathbf{X}^T)(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) \\ &= \mathbf{y}^T\mathbf{y} - \mathbf{y}^T\mathbf{X}\boldsymbol{\beta} - \boldsymbol{\beta}^T\mathbf{X}^T\mathbf{y} + \boldsymbol{\beta}^T\mathbf{X}^T\mathbf{X}\boldsymbol{\beta}. \end{aligned}$$

Notice that the following equality $\mathbf{y}^T\mathbf{X}\boldsymbol{\beta} = \boldsymbol{\beta}^T\mathbf{X}^T\mathbf{y}$ is always true. Indeed, the left term $\mathbf{y}^T\mathbf{X}\boldsymbol{\beta}$ is a product of matrices of dimensions $(1 \times n)(n \times m)(m \times 1)$ and is thus a scalar. However, a scalar is equal to its transposed, it means that it is equal to

$$(\mathbf{y}^T\mathbf{X}\boldsymbol{\beta})^T = \boldsymbol{\beta}^T\mathbf{X}^T(\mathbf{y}^T)^T = \boldsymbol{\beta}^T\mathbf{X}^T\mathbf{y}.$$

This allows to rewrite the function f as follows

$$f(\boldsymbol{\beta}) = \boldsymbol{\beta}^T\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} - 2\boldsymbol{\beta}^T\mathbf{X}^T\mathbf{y} + \mathbf{y}^T\mathbf{y}.$$

As we would like to minimise f compared to $\boldsymbol{\beta}$, we must equal the partial derivatives of this function to zero. For $i \in \{1, ..., m\}$, the partial derivative of f compared to $\boldsymbol{\beta}_i$ is computed as

$$\begin{aligned} \frac{\partial f}{\partial \boldsymbol{\beta}_i} &= \frac{\partial}{\partial \boldsymbol{\beta}_i}\Big(\sum_{k,l=1}^{m} \boldsymbol{\beta}_k\boldsymbol{\beta}_l(\mathbf{X}^T\mathbf{X})_{kl} - 2\sum_{k=1}^{m} \boldsymbol{\beta}_k(\mathbf{X}^T\mathbf{y})_k + \mathbf{y}^T\mathbf{y}\Big) \\ &= \sum_{l=1}^{m}(\mathbf{X}^T\mathbf{X})_{il}\boldsymbol{\beta}_l + \sum_{k=1}^{m}(\mathbf{X}^T\mathbf{X})_{ki}\boldsymbol{\beta}_k - 2(\mathbf{X}^T\mathbf{y})_i \\ &= 2\sum_{l=1}^{m}(\mathbf{X}^T\mathbf{X})_{il}\boldsymbol{\beta}_l - 2(\mathbf{X}^T\mathbf{y})_i \\ &= 2\big(\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} - \mathbf{X}^T\mathbf{y}\big)_i \end{aligned}$$

The fact that $\mathbf{X}^T\mathbf{X}$ is symmetric is used to go from the second to the third equality. Indeed, the multiplication of a matrix and its transposed is a symmetric matrix. Thus, by cancelling the term $\frac{\partial f}{\partial \boldsymbol{\beta}_i}$, we get

$$(\mathbf{X}^T\mathbf{X}\boldsymbol{\beta})_i = \left(\mathbf{X}^T\mathbf{y}\right)_i \qquad \forall i \in \{1, ..., m\}$$

this means

$$(\mathbf{X}^T\mathbf{X})\boldsymbol{\beta} = \mathbf{X}^T\mathbf{y}.$$

This can be rewritten as $\boldsymbol{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$ since the matrix $\mathbf{X}^T\mathbf{X}$ is of maximal rank. Finally, let us check that this is a minimum. We have

$$\frac{\partial^2 f}{\partial^2 \boldsymbol{\beta}_i} = \frac{\partial}{\partial \boldsymbol{\beta}_i}\, 2\left(\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} - \mathbf{X}^T\mathbf{y}\right)_i = 2(\mathbf{X}^T\mathbf{X})_i \quad \forall i \in \{1, ..., m\}$$

which implies that $\dfrac{\partial^2 f}{\partial \boldsymbol{\beta}\, \partial \boldsymbol{\beta}^T} = 2\mathbf{X}^T\mathbf{X}$. This matrix is positive-definite because it is of maximal rank and for $\mathbf{w} \in \mathbb{R}^m \setminus \{0\}$, we have

$$\mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w} = (\mathbf{X}\mathbf{w})^T(\mathbf{X}\mathbf{w}) = ||\mathbf{X}\mathbf{w}||^2 > 0.$$

Thus, as announced, the optimal searched vector $\widehat{\boldsymbol{\beta}}$ is obtained through

$$\widehat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}.$$

□

Least squares in machine learning

Using the same notations as above, given a dataset $\{\mathbf{x}_i, y_i\}_{i=1}^n$ composed of n observations, the algorithm assumes to have a linear relation between input data $\mathbf{x}_i$ and output data y_i, the responses. It uses the relation

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta}$$

to predict the value of the output variable $\mathbf{y}$. This relation is our model. During training, the prediction error of the output data must be minimised by finding the ideal value of $\boldsymbol{\beta}$. It is then necessary to have a function to compute this error. In machine learning, this function is called the **loss function** and it is used to measure the difference between the predicted value, $(\mathbf{X}\boldsymbol{\beta})_i$, and the real value, y_i. Let us take the quadratic loss function to compute the model error. It is defined by

$$L(\boldsymbol{\beta}) = \sum_{i=1}^{n}(y_i - (\mathbf{X}\boldsymbol{\beta})_i)^2 = ||\mathbf{y} - \mathbf{X}\boldsymbol{\beta}||^2.$$

This loss function must now be minimised, and this is where the (ordinary) least squares method is used to determine the coefficients of $\boldsymbol{\beta}$. Indeed, this procedure aims to minimise the sum of the squared residuals between the observed data and the values predicted by the linear approximation. Thus, this corresponds to compute the distance between each observation available in the data and its prediction, i.e. its projection on the regression line, to square them up and add up all these errors. Then, the least squares technique aims to minimise this total error quantity by fitting the model, which is the regression line.

Remark 2.1.2. We have just detailed an algorithm based on a linear model. However, only for very simple models a closed-form solution is known. For more complicated non-linear models, the minimum of the loss function is often approximated numerically by using the gradient of the loss function. In practise, there are many different algorithms that can be used for linear or non-linear applications. For further information, the course *"Introduction to Machine Learning"* [21] and the book [27] present different classical machine learning algorithms.

2.1.3 Mathematical concepts of machine learning

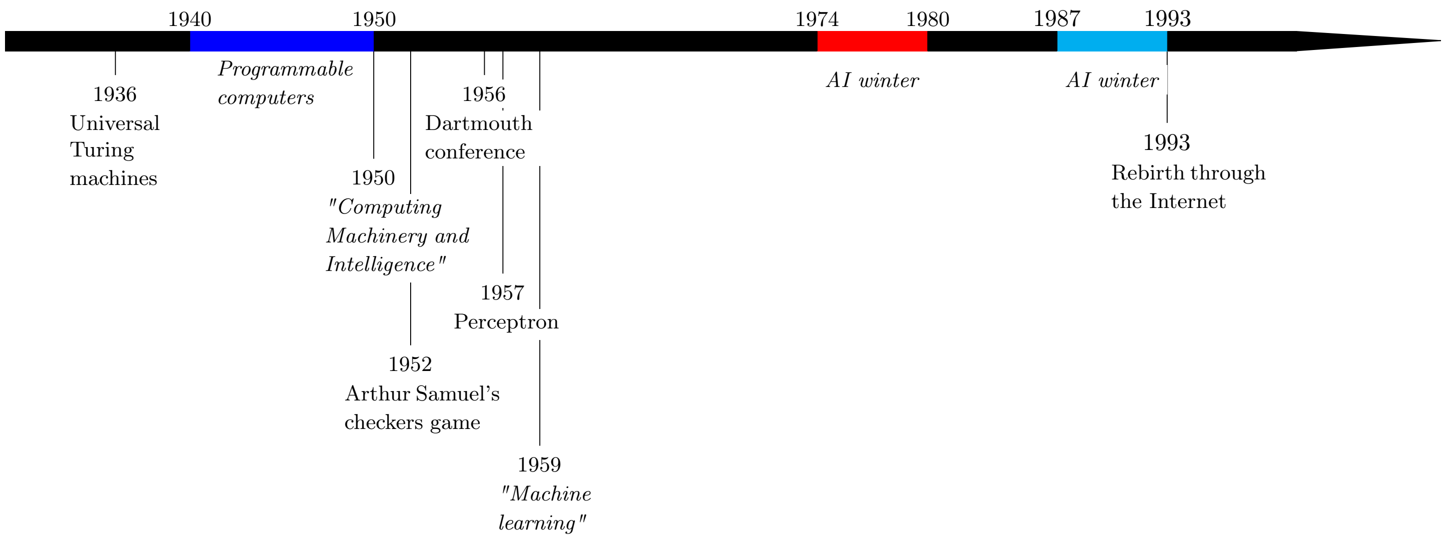
As explained previously, machine learning allows to create a model for automatically extracting information from data. To this end, many branches of mathematics are used. Indeed, linear algebra, probability, statistics, multivariate analysis and discrete mathematics are very useful in this field. This section aims to give an idea of the utility of these mathematical concepts in machine learning. To go further, references [13] and [32] detail these ideas.

Concepts of linear algebra are involved in the performance evaluation of a model and in the development of models. Matrix operations, LU and QR decompositions, eigenvalue and eigenvector search techniques, eigenspaces and many others are used in machine learning. When the dataset is huge, linear algebra can also be very useful. Probabilities are used for predicting future events or class labels. For instance, the maximum likelihood estimation technique, probabilities laws, mainly the normal law, joint, marginal and conditional probabilities are widely used in practice. Statistics allow to better understand the data and to draw conclusions from it. For example, principal component analysis, a dimension reduction technique, is very useful when the data available is of high dimension. When a model is optimised, it is often necessary to use vector functions and to compute partial derivatives, gradients, differentials or integrals. Finally, discrete mathematics also finds its application in machine learning. Indeed, graph structures are increasingly used in this field, hence the growing interest in graph theory. Several sub-fields of combinatorics can also be used.

It should be noted that people using machine learning do not, in practice, need to know every single mathematical detail mentioned above, the basics are enough. However, it is interesting for these people to investigate these mathematics in order to better understand the hidden workings behind the learning algorithms. Indeed, it is possible to use these pre-implemented algorithms available in the libraries of the software employed without knowing the internal mathematical details of the algorithm. Nevertheless, a better understanding of how the algorithm works allows better use of it and therefore better results in practice.

2.1.4 Machine learning history

Machine learning relies on many branches of mathematics, on advances in the design of a computer, on researches on the definition of intelligence, on the moment at which one can qualify a being of intelligent, on brain activity, on the functioning of human neurons and on its mathematical modelling. The timeline below indicates the main steps of the machine learning evolution covered in this section. The articles [6], [24] and [58], on which this section is based, can be consulted for more details on the chronology of events.


1940
1950
1974
1980
1987
1993
Programmable computers
AI winter
AI winter
1936 Universal Turing machines
1956 Dartmouth conference
1993 Rebirth through the Internet
1950 "Computing Machinery and Intelligence"
1957 Perceptron
1952 Arthur Samuel's checkers game
1959 "Machine learning"

The concept of *thinking machine* was employed for the first time by Alan Turing in his article *"Computing Machinery and Intelligence"* [57]. This paper mainly corresponds to the starting point of the field of artificial intelligence, which the beginning of machine learning is linked to. The following describes the computer science context in which this article was published, and the evolution of the field after this publication.

In 1936, Turing presented the concept of *universal Turing machine*, the concept at the basis of modern computers.[4] In the 1940s, there were already programmable computers able to retain instructions and data. However, Turing was the only one to imagine that this kind of machine could one day think and learn. In his 1950 paper, he introduced the *imitation game*[5], now called the *Turing test*, to determine whether a machine does what a being with the ability to think can do. Indeed, he thought a machine could be qualified as intelligent if it was able to pretend to be a human although it was a machine. In its article, Turing also introduced the idea behind a learning algorithm and of supervised learning, which was presented in Section 2.1.1.

Machine learning really started when, in 1952, the computer scientist Arthur Samuel wrote the first computer program for playing checkers with the ability to learn while playing. This program had a limited amount of available memory but ended up beating the fourth ranked player of the United States after its creator spent three years improving it. However, it was the Dartmouth Conference in 1956 that most intensely marked the beginning of machine learning. This conference followed a two-month study involving 11 well-known researchers in the fields of mathematics, engineering, computer science and cognitive science, including Marvin Minsky, John McCarthy, Ray Solomonoff and Claude Shannon. The aim of the study was to investigate the characteristics of human learning and human intelligence so that a machine could replicate it, to determine how a machine could use a given language and to solve some kinds of problems for which humans are able

[4]They are an improvement of the Turing Machines, which are simple machines that can read symbols from paper, have an internal memory and can write symbols on paper.

[5]Initially, this game was played by humans and the initial goal of this game for a woman player was to be able to fool an interlocutor into thinking she is a man and not a woman, and inversely for a man. Turing thought that a machine could be called intelligent if it was able to do the same things as a human, i.e. to pretend to be a human although it was a machine.

to find a solution and improve themselves.[6] It was at this conference that the term *artificial intelligence* was introduced by John McCarthy. This study helped motivate researchers at that time to build a machine capable of thinking and learning. In 1959, Arthur Samuel used the term *machine learning* for the first time for the program he had created in 1952.

During the 1950s and 1960s, this field was very popular and many algorithms were created. For example, Frank Rosenblatt created the *perceptron* in 1957, which allows a classification into two groups of a given data set. However, following many failures, researchers turned away from the field, pessimistic about its effectiveness, funding collapsed and what is known as the "winter of AI" began in 1974. This field attracted again in the 1980s but went through a second so-called "AI winter" from the end of the 1980s to the beginning of the 1990s. The second rebirth of the field was mainly due to the increase in computing power of computers. The emergence of the Internet, allowing researchers to communicate more easily and quickly with each other about their results and to have access to many large databases, has also led to huge advances in the field since machine learning tends to work better on large data sets.

This field has gone from planning routes (the traveller's problem) or from playing not too complex board games, such as checkers or tic-tac-toe, to voice recognition, image classification or even a chess player better than the human world champion. Nowadays, machine learning is used in many different fields such as video games, search systems, autonomous cars, medicine or even identification of human faces in an image.

Deep learning is widely used for speech recognition and image processing. As its basic principle is to mimic human neural networks, it theoretically began in 1943, when a model based on neural networks was created. Afterwards, its development more or less followed that of machine learning. *Computer vision*, on the other hand, emerged a little later, in the 1960s. Indeed, some progress were required in computer science for a program to become efficient enough at object detection. It is also due to the arrival of the Internet, providing large datasets containing huge amounts of images, and the performance of today's computers that this field is very widespread nowadays.

2.2 Bases of deep learning

Let us now take a closer look at deep learning. Remember that we are working with images of ships, and that the model should detect these ships. For this purpose, an input to a model will be an image. An image is represented by a matrix of numbers, but in the case of the simple model structure presented below, this matrix is transformed into a vector. We therefore note this input $\mathbf{x} \in \mathbb{R}^{n_1}$.[7] It will then be fed to the chosen model with some parameter $\mathbf{W}$, called the **weight matrix**, and the bias $\boldsymbol{b}$.

We note the output of the model as $\mathbf{y} = f(\mathbf{x}; \mathbf{W}, \boldsymbol{b})$.[8] In our case, the output $\mathbf{y}$ will

[6]It was in August 1955 that John McCarthy and the organisers proposed the following project: *"We propose that a 2 month, 10 man study of artificial intelligence be carried out during the summer of 1956 at Dartmouth College in Hanover, New Hampshire. The study is to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it. An attempt will be made to find how to make machines use language, form abstractions and concepts, solve kinds of problems now reserved for humans, and improve themselves."*

[7]The variable n_1 is the number of nodes in the input layer, these terms will be introduced in this section.

[8]For instance, in Section 2.1.2, the parameter was the vector $\boldsymbol{\beta}$ of unknown values and the model was written as $\mathbf{y} = \mathbf{X}\boldsymbol{\beta}$.

correspond to a list of positions of the detected ships that was present in the input image, but in general **y** can be of various forms, such as a set of numbers or an image. Recall that the model improves itself by adjusting the values of the parameters **W** and ***b*** during the training step in order to minimise the loss function which measures the prediction error of the model. This function has briefly been explained previously, but will be presented in more detail later.

In this part, we will first introduce in Section 2.2.1 the typical deep learning structure of a model producing the function f, then in Section 2.2.2, the loss function and in Section 2.2.3, the backpropagation algorithm, used to optimise the weights. Section 2.2.4 will explain what is the learning sequence and finally, Section 2.2.5 will present convolutional neural network, which is a type of deep learning network that will be used a lot throughout this work.

This part is mainly based on the material covered in the *"Introduction to machine learning"* [21] and *"Deep learning"* [37] courses, but other references were also used and, if so, are indicated at the beginning of the corresponding section.

2.2.1 Neural networks structure

In the introduction of this document, deep learning was presented as a branch of machine learning, which uses a particular algorithm, called a **(artificial) neural network**, to mimic the behaviour of the human brain. For this purpose, neural networks attempt to mimic neurons. In the brain, these are connected to each other. Each neuron receives several input signals which it then treats in order to provide an output signal based on the received input signals.

In deep learning, the **nodes** in the neural network aim to imitate the neurons. As such, they will be connected together. Figure 2.1 illustrates the structure of a node.

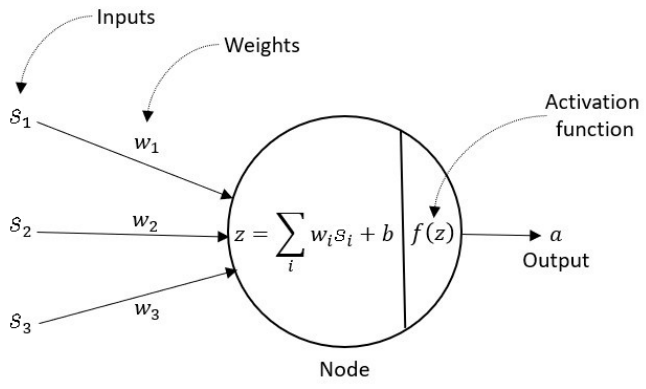


Figure 2.1: Structure of a node in a neural network with 3 inputs and a single output. © [1]

The structure of a given node is analysed as follows:

- As biological neurons, a node receives several inputs, noted as $s_i \in \mathbb{R}$ for $i \in \{1, ..., n\}$ where n is the number of inputs. These values come either from the network inputs, or from the outputs of previous nodes in the network, or a combination of both.
- Each input s_i of the node is weighted by a value $w_i \in \mathbb{R}$, which results in $w_i s_i$.

- These weighted values are then summed together with the bias $b \in \mathbb{R}$,[9] which results in
$$\sum_{i=1}^{n} w_i s_i + b = \mathbf{w}^T \mathbf{s} + b \qquad \text{for } i \in \{1, ..., n\}$$
where $\mathbf{w} = (w_1, ..., w_n)^T$ and $\mathbf{s} = (s_1, ..., s_n)^T$.

- An **activation function** is then applied to this sum to activate the node or not, whether its input is important or not. If it has been activated, this function transforms this sum into some output to be fed to the next layer. A non-linear activation function can be used to get rid of the linearity of the term $\mathbf{w}^T\mathbf{s}+b$.[10] The application of this function is noted as
$$a = \varphi(\sum_{i=1}^{n} w_i s_i + b) = \varphi(\mathbf{w}^T \mathbf{s} + b),$$
where φ is the activation function.[11] For instance, some classic examples of activation function are the sigmoid function defined as
$$\sigma(z) = \frac{1}{1 + \exp(-z)} \quad \text{for } z \in \mathbb{R},$$
the hyperbolic tangent defined as
$$\tanh(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)} \quad \text{for } z \in \mathbb{R},$$
which is just a rescaled sigmoid since $\tanh(z) = 2\sigma(2z) - 1$, and the rectified linear unit (ReLU) defined as
$$f(z) = \max(0, z) \quad \text{for } z \in \mathbb{R}\,.$$

- As biological neurons, a node will produce an output a, which will then be used as input in the following nodes.

As said earlier, nodes are interconnected. Actually, in neural networks, they are grouped into layers, which are connected to each other. However, nodes of the same layer are not interconnected. There are three main types of layers.

[9]Without going into details, the bias term, which is linear with respect to $\mathbf{s}$, allows to increase the ability of the model to fit the data. This provides an output that is linear with respect to the inputs.

[10]The following briefly explains why it is interesting to destruct linearity. A first reason is that it reduces the risk of having a very large output value of the node. Indeed, without an activation function, the output of the node is $\mathbf{w}^T\mathbf{s} + b$, where the bias term, b, can be very large in neural networks that have millions of parameters. This implies computational problems. Then, the application of a non-linear activation function prevents the output value of the node from exploding.

A second reason for using such functions is to handle more complex problems. In practice, it is not possible to approach them using only nodes producing a linear output. Thus, the addition of non-linear activation functions allows for a higher degree of complexity since it adds non-linearity to the neural network by adding it at the node output. [25]

[11]From a mathematical point of view, the activation function must be differentiable for the backpropagation algorithm, which will be presented in Section 2.2.3. However, in practice, an activation function that is differentiable almost everywhere is sufficient because these points are rarely reached and solutions are implemented otherwise. Consider the example of ReLu, which is not differentiable at 0. If the gradient at 0 has to be calculated, depending on the implementation used, the derivative at this point is directly replaced by 0, see [25] and [37].

1. **Input layer:** each node of this layer is fed with one input and outputs one value, which will then be fed to one or more nodes in the next layer. Inputs of this layer correspond to the inputs of the network. Thus, each node is fed by a component of the input vector $\mathbf{x} \in \mathbb{R}^{n_1}$, where n_1 indicates the number of nodes in this layer.

2. **Hidden layer:** in the neural network, there may be one or more layers of this type. Each node of these layers is fed with one or several values from the previous layer and outputs one value, which will then be fed to one or more nodes in the next layer. This kind of layer is useful to create features based on the received inputs.

3. **Output layer:** each node of this layer is fed with one or several values coming from the last hidden layer and outputs one value. For example, in a regression problem where the output of the network should be a single numerical value, there is only one node in the output layer that predicts this value, but if multiple numerical values are expected, then there are multiple nodes in the output layer. On the other hand, if the network should perform a classification task, then there are c nodes in this layer, where c is the number of classes.

Neural networks are composed of one input layer, at least one hidden layer and one output layer. Figure 2.2 illustrates the structure of a neural network with 3 inputs, 2 hidden layers with 4 nodes each and 2 outputs. This network is said to be **fully connected** because each node in each layer is connected to each node in the next layer.

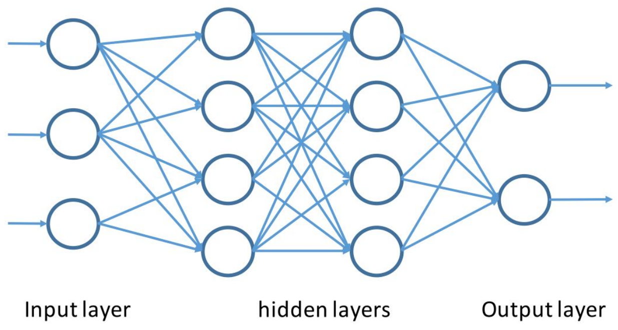


Figure 2.2: Structure of a fully connected neural network with 3 inputs, 2 hidden layers with 4 nodes each and 2 outputs. © [21]

Next, we introduce some notations on this architecture. Suppose that the neural network has L layers, where layer 1 is the input layer, layers from 2 to $L-1$ are the hidden layers and layer L is the output layer. For each layer $l \in \{1, ..., L\}$, we note

- n_l: the number of nodes in layer l, where n_1 is the number of inputs and n_L the number of outputs of the network;
- $\boldsymbol{W^{(l)}}$: the weight matrix between layers $l-1$ and l, where $w_{k,j}^{(l)}$ is the weight of the edge from node j in layer $l-1$ to node k in layer l, for $l \geq 2$, $j \in \{1, ..., n_{l-1}\}$ and $k \in \{1, ..., n_l\}$;
- $\boldsymbol{b^{(l)}}$: the vector of biases in layer l, where $b_k^{(l)}$ is the bias of node k for $k \in \{1, ..., n_l\}$;
- $\varphi^{(l)}$: the activation function of layer l;
- $\boldsymbol{a^{(l)}}$: the vector of the activations in layer l, where $a_j^{(l)}$ corresponds to the output of the j^{th} node, for $j \in \{1, ..., n_l\}$.

By setting $\boldsymbol{a^{(0)}} = \mathbf{x}$ and $\boldsymbol{W^{(1)}}, \boldsymbol{b^{(1)}}$ to the initial values of the model parameters $\mathbf{W}, \boldsymbol{b}$, the vector of activations $\boldsymbol{a^{(l)}}$ of layer l can be computed successively from layer 1 to layer L using

$$\begin{cases} \boldsymbol{z^{(l)}} = \boldsymbol{W^{(l)}}\boldsymbol{a^{(l-1)}} + \boldsymbol{b^{(l)}} \\ \boldsymbol{a^{(l)}} = \varphi^{(l)}(\boldsymbol{z^{(l)}}) \end{cases}, \forall\, l \in \{1, ..., L\}.$$

Finally, the vector of activations $\boldsymbol{a^{(L)}}$ in layer L provides the output $\hat{\mathbf{y}} = f(\mathbf{x}; \mathbf{W}, \boldsymbol{b})$. Computing these values successively from the input layer to the output layer is called the **forward pass**.

2.2.2 Loss function

We briefly explained what the loss function is on the example of the method of least squares in Section 2.1.2 and recalled in this section that it was used to adjust the model parameters. Now, we go a bit further. In addition to the courses [21] and [37], this section is also partly based on [18] and [39].

The **loss function** is computed at each step of the model training to measure the errors in its predictions. Computing this at each step allows to see if the model makes less and less prediction errors with time, i.e. that better solutions to the optimisation problem are found. Before defining this more formally, some notations are introduced.

- The set of training data we are working with is noted as

$$\{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), ..., (\mathbf{x}_n, \mathbf{y}_n)\}.$$

  There are n pairs of observations, where, for $i \in \{1, ..., n\}$,

  - $\mathbf{x}_i \in \mathbb{R}^p$ is an input variable, which is a p-dimensional vector of features,
  - $\mathbf{y}_i \in \mathbb{R}^m$ is its associated outcome, which is a scalar representing the probability to belong to a certain category for a classification problem, or is a m-dimensional vector of real values for a regression problem.

- We consider the function $f(\,\cdot\,; \boldsymbol{\theta}) : \mathbb{R}^p \to \mathbb{R}^m$ produced by the chosen learning algorithm. It depends on the model parameter $\boldsymbol{\theta}$ that represents the weights and the biases. This function takes as input the components $\mathbf{x}_i$ and returns the approximation of its associated value $\mathbf{y}_i$.

The **loss function** $\ell : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ aims to evaluate the predictions of function f such that $\ell(\mathbf{y}_i, f(\mathbf{x}_i; \boldsymbol{\theta}))$ measures how close the prediction of the neural network $f(\mathbf{x}_i; \boldsymbol{\theta})$ is from $\mathbf{y}_i$, $i \in \{1, ..., n\}$. In other words, this function compares the observations to the predictions made by the model. So, it tells the quality of the current classifier.

Above, the loss function has just been defined for a single prediction $\mathbf{y}_i$. However, it should take into account all predictions. The loss

$$L(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} \ell(\mathbf{y}_i, f(\mathbf{x}_i; \boldsymbol{\theta})) \tag{2.1}$$

corresponds to the average loss over the training data, i.e. the loss of each input variable prediction is now taken into account in the global loss L. This loss takes $\boldsymbol{\theta}$ as argument since this term represents the parameters of the model that the latter uses to make predictions.

One must choose which loss function to use. There exist several functions that can be used for this, such as the quadratic loss function used in the least squares example, see Section 2.1.2, which is defined as

$$\ell(\mathbf{y}_i, f(\mathbf{x}_i; \boldsymbol{\theta})) = ||\mathbf{y}_i - f(\mathbf{x}_i; \boldsymbol{\theta})||^2$$

and leads to the following average loss, called the mean squared error (MSE),

$$L(\boldsymbol{\theta}) = \mathrm{MSE}(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} ||\mathbf{y}_i - f(\mathbf{x}_i; \boldsymbol{\theta})||^2.$$

This is the most common choice for regression, while it is the cross-entropy loss for classification. This is defined as

$$\ell(\mathbf{y}_i, f(\mathbf{x}_i; \boldsymbol{\theta})) = -\sum_{c=1}^{M} t_{\mathbf{x}_i,c} \log(p_{\mathbf{x}_i,c}),$$

where M is the number of classes, $t_{\mathbf{x}_i,c} = 1$ if $\mathbf{x}_i$ belongs to class c, 0 otherwise; and $p_{\mathbf{x}_i,c}$ is the probability that $\mathbf{x}_i$ belongs to class c. This leads to the average loss

$$L(\boldsymbol{\theta}) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{c=1}^{M} t_{\mathbf{x}_i,c} \log(p_{\mathbf{x}_i,c}).$$

The choice of loss function therefore depends on the problem, whether it is a classification or a regression, and the functions listed above have some drawbacks so, depending on the problem, other loss functions may be more appropriate. For example, the MSE loss function penalises outliers too much by squaring them and the cross-entropy loss is sensitive to imbalanced data.

At this point, the loss function L has been defined and it can measure the prediction error. It is now important to understand how this function is used in order to adjust the parameters of the model for it to make better predictions.

We are in fact facing an optimisation problem. Indeed, the prediction error should be as small as possible for each observation of the training data. So, the loss function (2.1) should be minimised. The only thing that can vary in the model during training is the parameter $\boldsymbol{\theta}$. Thus, we would like to solve the following minimisation problem

$$\arg\min_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \frac{1}{n}\sum_{i=1}^{n} \ell(\mathbf{y}_i, f(\mathbf{x}_i; \boldsymbol{\theta}))$$

This is the final goal at the end of all learning steps. At each of them, the objective is to update $\boldsymbol{\theta}$ to decrease the loss value. The following explains how this parameter is modified at each step with the aim of minimising the loss during the next few learning steps.

For this purpose, the **gradient descent** method is used. This is a general minimisation technique to find numerically a minimiser. It uses local linear information to iteratively move towards a (local) minimum. If $\boldsymbol{\theta}_0$ denotes the initial values of the model parameters, a first-order approximation around $\boldsymbol{\theta}_0$ can be defined, using the Taylor approximation,[12] as

$$\hat{L}(\boldsymbol{\theta}_0 + \boldsymbol{\epsilon}) = L(\boldsymbol{\theta}_0) + \boldsymbol{\epsilon}^T \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_0) + \frac{1}{2\gamma}||\boldsymbol{\epsilon}||^2 \quad \text{with } \boldsymbol{\epsilon} \to 0,$$

[12]See Appendix A for more information about the derivation of this expression.

where $\nabla_{\mathbf{x}} g(\mathbf{x})$ indicates the gradient of a function g with respect to $\mathbf{x}$, i.e. the vector of partial derivatives

$$\nabla_{\mathbf{x}} g(\mathbf{x}) = \begin{pmatrix} \frac{\partial g}{\partial x_1}(\mathbf{x}) \\ \frac{\partial g}{\partial x_2}(\mathbf{x}) \\ \vdots \\ \frac{\partial g}{\partial x_k}(\mathbf{x}) \end{pmatrix} \qquad \text{where } \mathbf{x} = (x_1, x_2, ..., x_k)^T$$

and where $\gamma > 0$ is the learning rate. The **learning rate** is a coefficient that determines the size of each learning step while getting closer to the minimum of the loss function. The approximation is minimised when

$$\begin{aligned} &\nabla_{\boldsymbol{\epsilon}} \hat{L}(\boldsymbol{\theta}_0 + \boldsymbol{\epsilon}) = 0 \\ \Leftrightarrow\ &\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_0) + \frac{1}{\gamma}\boldsymbol{\epsilon} = 0 \\ \Leftrightarrow\ &\boldsymbol{\epsilon} = -\gamma \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_0) \end{aligned}$$

Thus, the best improvement for this step is $\boldsymbol{\epsilon} = -\gamma \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_0)$. So, at the next learning step, the parameter is updated as $\boldsymbol{\theta}_1 = \boldsymbol{\theta}_0 - \gamma \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_0)$ and the loss can be computed using this new value.

The same process is applied for the next steps and at the learning step $t+1$, the model parameters have been updated iteratively using the following rule

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \gamma \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t), \tag{2.2}$$

with $\boldsymbol{\theta}_t$ the value of the model parameters at step t. Note that the convergence of the optimisation problem using the update rule is very much dependent on the choice of the initial parameter $\boldsymbol{\theta}_0$ and of the learning rate γ.

The optimisation process stops either after a fixed number of steps, or when the value of the loss function has become smaller than a given threshold.

For a given learning step with model parameter $\boldsymbol{\theta}$, the term

$$\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} \nabla_{\boldsymbol{\theta}} \ell(\mathbf{y}_i, f(\mathbf{x}_i; \boldsymbol{\theta}))$$

has to be computed. The complexity of this computation grows linearly with the size n of the dataset. This is a disadvantage of this version of gradient descent, called **batch gradient descent**. In this case, a **batch** corresponds to the complete training set and, at each learning step, the parameters are updated based on all the samples included in this batch.

Another version of gradient descent, **stochastic gradient descent**, does not have this drawback and uses instead the following update rule

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \gamma \nabla_{\boldsymbol{\theta}} \ell(\mathbf{y}_{i(t+1)}, f(\mathbf{x}_{i(t+1)}; \boldsymbol{\theta}_t)), \tag{2.3}$$

where the samples $i(t)$ are selected randomly at each iteration. The complexity of this computation is now independent of n. Another advantage of this version is that there is less risk of getting stuck in a local minima during the optimisation process. Indeed, it is easier to escape from it thanks to the randomness introduced by this version when updating the parameters. However, one drawback is that the value of the loss function does not necessarily decrease at each step, as a single sample is used for the update, so this version is slower to converge, hence **mini-batch gradient descent**, which is a combination of these versions, is generally preferred. Instead of using all the samples, i.e. the batch, or a single sample from the training set to update the parameters, the batch is divided into several subsets, called **mini-batches**. At each learning step, the parameters are then updated based on the samples included in a mini-batch. In this case, the update rule is defined as

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \gamma \frac{1}{B} \sum_{b=1}^{B} \nabla_{\boldsymbol{\theta}} \ell(\mathbf{y}_{i(t,b)}, f(\mathbf{x}_{i(t,b)}; \boldsymbol{\theta}_t)), \tag{2.4}$$

where B is the size of the mini-batches and the order $i(t, b)$ to visit the samples can be random or sequential.

For each gradient descent version, the update rule can be calculated efficiently through **backpropagation**.

2.2.3 Backpropagation

As discussed in the previous section, several gradient descent methods are available for adjusting the model parameters to minimise the loss function. For this purpose, these methods require the gradient of this function with respect to the model parameters to be computed. **Backpropagation** is one method for efficiently computing it.[13] For instance, using stochastic gradient descent with backpropagation is one way to train a neural network. In addition to the courses [21] and [37], this section is also based on [18].

When one gradient descent method with backpropagation is selected, several forward and backward passes through the network are performed. In other words, the following steps are repeated a certain number of times, this being the number of **epochs**, or until the loss function value becomes smaller than some threshold ϵ.

1. **Forward pass** (or **forward propagation**): all values from the input layer are propagated through the hidden layers to the output layer. The resulting values correspond to the network predictions, which are then used to compute the value of the loss function. The details of the computation of this pass have already been described at the end of Section 2.2.1.

2. **Backward pass** (or **backward propagation** or **backpropagation**): the error term computed in the previous step is propagated in the other direction, i.e. from the output layer through the hidden layers to the input layer. Now, the gradient of the loss function with respect to the parameters can be computed. Once this is done, the parameters of the model, the weights and the biases, are adjusted based on the update rule of the selected gradient descent method, see Equations (2.2), (2.3) or (2.4). Further details are provided below. Finally, as the parameters have been updated, a new forward pass can be performed, which should lead to a loss function value smaller than the previous one.

[13]This is an efficient method since it does not naively compute the gradient with respect to each parameter in a separate way.

Figure 2.3 illustrates the directions of the forward and backward passes on a neural network architecture. In next sections, a "pass through the network" indicates that a forward pass and a backward pass have been completed. In other words, it corresponds to an estimation of the gradient and an update of the model parameters.

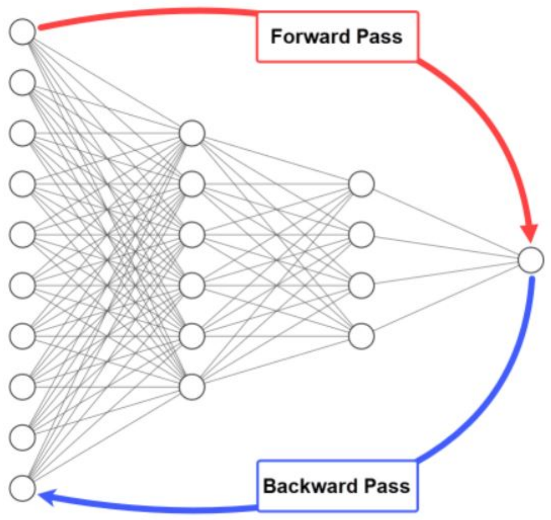


Figure 2.3: Illustration of the directions of the forward and backward passes. © [19]

In summary, the backpropagation algorithm used with a chosen gradient descent method enables the computation of the derivatives of the loss function with respect to the model parameters. For this purpose, two passes are performed, i.e. the forward and backward pass. Some mathematical details on how these passes work will now be provided.

First, recall that with stochastic gradient descent, it is sufficient to compute the per-sample loss with respect to the parameter $\boldsymbol{\theta}$, i.e. the loss associated to a single prediction. Indeed, the update rule of the parameters in this method is given by

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \gamma \nabla_{\boldsymbol{\theta}} \ell(\mathbf{y}_{i(t+1)}, f(\mathbf{x}_{i(t+1)}; \boldsymbol{\theta}_t)),$$

using the same notations as in Section 2.2.2. In particular, recall that the samples $i(t)$ are selected randomly at each iteration. Hence, the goal is to compute $\nabla_{\boldsymbol{\theta}} \ell(\mathbf{y}_i, f(\mathbf{x}_i; \boldsymbol{\theta}_t))$ for $i \in \{1, ..., n\}$ with respect to $\boldsymbol{\theta} = (\mathbf{W}, \boldsymbol{b})$. As it is sufficient to compute the per-sample loss, we consider a single training sample $\mathbf{x}$. Thus, the associated loss function we are working with is $\ell(\mathbf{y}, f(\mathbf{x}; \boldsymbol{\theta})) = \ell(\mathbf{y}, \hat{\mathbf{y}})$. In this section, the evaluation of the loss function in values $\mathbf{y}$ and $\hat{\mathbf{y}}$ has been removed to simplify the notation.

To summarise, the goal is to compute the following partial derivatives for each layer $l \in \{1, ..., L\}$,

$$\frac{\partial \ell}{\partial w_{k,j}^{(l)}} \quad \text{and} \quad \frac{\partial \ell}{\partial b_k^{(l)}} \quad \forall k \in \{1, ..., n_l\} \text{ and } \forall j \in \{1, ..., n_{l-1}\}. \tag{2.5}$$

The notations used for the parameters are the same as those presented at the end of Section 2.2.1.

Forward pass

The details of this pass have already been presented previously, but are summarised below.

$$\boldsymbol{a}^{(0)} = \mathbf{x} \quad \text{and} \quad \forall\, l \in \{1, ..., L\}, \begin{cases} \boldsymbol{z}^{(l)} = \boldsymbol{W}^{(l)}\boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)} \\ \boldsymbol{a}^{(l)} = \varphi^{(l)}(\boldsymbol{z}^{(l)}) \end{cases}$$
$$\text{and} \quad \boldsymbol{a}^{(L)} = f(\mathbf{x}; \mathbf{W}, \boldsymbol{b}) = \hat{\mathbf{y}},$$

where $\varphi^{(l)}$ denotes the activation function in layer l. Successively computing the values $\boldsymbol{z}^{(l)}$ and their corresponding activation $\boldsymbol{a}^{(l)}$ for each layer $l \in \{1, \ldots, L\}$ corresponds to the forward pass. At the end, the prediction $\boldsymbol{a}^{(L)} = f(\mathbf{x}; \mathbf{W}, \boldsymbol{b}) = \hat{\mathbf{y}}$ is available and the loss function can be computed. The computation of this pass is represented as follows

$$\boldsymbol{a}^{(0)} = \mathbf{x} \xrightarrow{\boldsymbol{W}^{(1)}, \boldsymbol{b}^{(1)}} \boldsymbol{z}^{(1)} \xrightarrow{\varphi^{(1)}} \boldsymbol{a}^{(1)} \xrightarrow{\boldsymbol{W}^{(2)}, \boldsymbol{b}^{(2)}} \boldsymbol{z}^{(2)} \xrightarrow{\varphi^{(2)}} \boldsymbol{a}^{(2)} \longrightarrow \ldots \xrightarrow{\boldsymbol{W}^{(L)}, \boldsymbol{b}^{(L)}} \boldsymbol{z}^{(L)} \xrightarrow{\varphi^{(L)}} \boldsymbol{a}^{(L)} = \hat{\mathbf{y}}$$

Backward pass

To compute the backward pass, the following remark about the chain rule will be useful.

Remark 2.2.1 (Chain rule). Given a function $f : \mathbb{R}^n \to \mathbb{R}^m$ that decomposes as a chain composition

$$f = f_t \circ f_{t-1} \circ \ldots \circ f_1,$$

for functions $f_k : \mathbb{R}^{n_{k-1}} \to \mathbb{R}^{n_k}$ for $k \in \{1, ..., t\}$ with $n_0 = n$ and $n_t = m$, and assuming that

$$\mathbf{x}_0 = \mathbf{x} \text{ and } \mathbf{x}_k = f_k(\mathbf{x}_{k-1}) \;\; \forall k \in \{1, ..., t\},$$

where $\mathbf{x} \in \mathbb{R}^n$, these relations can be represented as

$$\mathbf{x}_0 \longrightarrow f_1 \longrightarrow \mathbf{x}_1 \longrightarrow f_2 \longrightarrow \ldots \longrightarrow \mathbf{x}_{t-1} \longrightarrow f_t \longrightarrow \mathbf{x}_t.$$

The chain rule provides the following equality

$$\frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_0} = \frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_{t-1}} \frac{\partial \mathbf{x}_{t-1}}{\partial \mathbf{x}_0}.$$

By applying this recursively, this can be rewritten as

$$\frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_0} = \frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_{t-1}} \frac{\partial \mathbf{x}_{t-1}}{\partial \mathbf{x}_{t-2}} \cdots \frac{\partial \mathbf{x}_2}{\partial \mathbf{x}_1} \frac{\partial \mathbf{x}_1}{\partial \mathbf{x}_0}.$$

The objective of the backward pass is to compute for each layer $l \in \{1, ..., L\}$ the gradient of the loss function $\ell(\mathbf{y}, \hat{\mathbf{y}})$ with respect to the parameters, i.e. to compute the derivatives shown in (2.5). First, we start by computing this for the output layer L, i.e. $\frac{\partial \ell}{\partial w_{k,j}^{(L)}}$ and $\frac{\partial \ell}{\partial b_k^{(L)}}$. We therefore work with the following relationship

$$\boldsymbol{a}^{(L-1)} \xrightarrow{\boldsymbol{W}^{(L)}, \boldsymbol{b}^{(L)}} \boldsymbol{z}^{(L)} \xrightarrow{\varphi^{(L)}} \boldsymbol{a}^{(L)} = \hat{\mathbf{y}}.$$

As $\boldsymbol{z}^{(L)} = \boldsymbol{W}^{(L)}\boldsymbol{a}^{(L-1)} + \boldsymbol{b}^{(L)}$, the parameter $\boldsymbol{b}^{(L)}$ influences the loss only through $\boldsymbol{z}^{(L)}$, so the chain rule can be applied, which provides

$$\frac{\partial \ell}{\partial b_k^{(L)}} = \frac{\partial \ell}{\partial z_k^{(L)}} \frac{\partial z_k^{(L)}}{\partial b_k^{(L)}} = \frac{\partial \ell}{\partial z_k^{(L)}} \qquad \text{for } k \in \{1, ..., n_L\}.$$

The second equality derives from $z_k^{(L)} = (\boldsymbol{W^{(L)}a^{(L-1)}})_k + b_k^{(L)}$ and therefore $\frac{\partial z_k^{(L)}}{\partial b_k^{(L)}} = 1$. Similarly, the parameter $\boldsymbol{W^{(L)}}$ influences the loss only through $\boldsymbol{z^{(L)}}$. Hence, by applying the chain rule, we get,

$$\frac{\partial \ell}{\partial w_{k,j}^{(L)}} = \frac{\partial \ell}{\partial z_k^{(L)}} \frac{\partial z_k^{(L)}}{\partial w_{k,j}^{(L)}} = \frac{\partial \ell}{\partial z_k^{(L)}} a_j^{(L-1)} \quad \text{for } k \in \{1, ..., n_L\} \text{ and } j \in \{1, ..., n_{L-1}\}$$

where $\frac{\partial z_k^{(L)}}{\partial w_{k,j}^{(L)}} = a_j^{(L-1)}$ since $z_k^{(L)} = \sum_{j=1}^{n_{L-1}} w_{k,j}^{(L)} a_j^{(L-1)} + b_k^{(L)}$.

This works similarly for the other layers, from layer $L-1$ to layer 1. Indeed, at each step, we are working with the following relationship

$$\boldsymbol{a^{(l-1)}} \xrightarrow{\boldsymbol{W^{(l)}},\, \boldsymbol{b^{(l)}}} \boldsymbol{z^{(l)}} \xrightarrow{\varphi^{(l)}} \boldsymbol{a^{(l)}}$$

and the parameters $\boldsymbol{b^{(l)}}$ and $\boldsymbol{W^{(l)}}$ influence the loss only through $\boldsymbol{z^{(l)}}$. Hence, by the chain rule, we get

$$\frac{\partial \ell}{\partial b_k^{(l)}} = \frac{\partial \ell}{\partial z_k^{(l)}} \quad \text{and} \quad \frac{\partial \ell}{\partial w_{k,j}^{(l)}} = \frac{\partial \ell}{\partial z_k^{(l)}} a_j^{(l-1)},$$

where $k \in \{1, ..., n_l\}$ and $j \in \{1, ..., n_{l-1}\}$.

However, to compute these partial derivatives, it is necessary to calculate $\frac{\partial \ell}{\partial z_k^{(l)}}$ for each layer l. Since $\boldsymbol{a^{(l)}} = \varphi^{(l)}(\boldsymbol{z^{(l)}})$, the variable $\boldsymbol{z^{(l)}}$ influences the loss function only through $\boldsymbol{a^{(l)}}$. The chain rule provides

$$\frac{\partial \ell}{\partial z_k^{(l)}} = \frac{\partial \ell}{\partial a_k^{(l)}} \frac{\partial a_k^{(l)}}{\partial z_k^{(l)}} = \frac{\partial \ell}{\partial a_k^{(l)}} \left(\varphi^{(l)}(z_k^{(l)})\right)',$$

where $\frac{\partial \ell}{\partial a_k^{(l)}}$ should have been computed in the previous step. It is therefore necessary to compute $\frac{\partial \ell}{\partial a^{(l-1)}}$ for the next step, i.e. for layer $l-1$.

Since $z_k^{(l)} = \sum_{j=1}^{n_{l-1}} w_{k,j}^{(l)} a_j^{(l-1)} + b_k^{(l)}$, the value $a_j^{(l-1)}$ influences the loss only through $z_k^{(l)}$, for $j \in \{1, ..., n_{l-1}\}$ and for all $k \in \{1, ..., n_l\}$. Hence, by the chain rule, we get

$$\frac{\partial \ell}{\partial a_j^{(l-1)}} = \sum_{k=1}^{n_l} \frac{\partial \ell}{\partial z_k^{(l)}} \frac{\partial z_k^{(l)}}{\partial a_j^{(l-1)}} = \sum_{k=1}^{n_l} \frac{\partial \ell}{\partial z_k^{(l)}} w_{k,j}^{(l)}.$$

In summary, the following steps allow to compute the backward pass.

1. In layer L, the value of $\frac{\partial \ell}{\partial a_k^{(L)}}$ can be computed directly from the definition of the loss function $\ell(\mathbf{y}, \hat{\mathbf{y}})$ since $a_k^{(L)} = \hat{y}_k$ for $k \in \{1, ..., n_L\}$.
2. For layer l from L to 1, iteratively compute, for $j \in \{1, ..., n_{l-1}\}$ and $k \in \{1, ..., n_l\}$:
   - $\frac{\partial \ell}{\partial z_k^{(l)}} = \frac{\partial \ell}{\partial a_k^{(l)}} \left(\varphi^{(l)}(z_k^{(l)})\right)',$

- $\dfrac{\partial \ell}{\partial a_j^{(l-1)}} = \sum\limits_{k=1}^{n_{l+1}} \dfrac{\partial \ell}{\partial z_k^{(l)}} w_{k,j}^{(l)}$,
- $\dfrac{\partial \ell}{\partial w_{k,j}^{(l)}} = \dfrac{\partial \ell}{\partial z_k^{(l)}} a_j^{(l-1)}$,
- $\dfrac{\partial \ell}{\partial b_k^{(l)}} = \dfrac{\partial \ell}{\partial z_k^{(l)}}$.

The values of the forward pass must be kept in memory to compute these derivatives. The memory cost of this algorithm is therefore high.

Now that the way the backpropagation algorithm computes the gradient of the loss function has been seen, we return to the stochastic gradient descent method. Its update rule was

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \gamma \nabla_{\boldsymbol{\theta}} \ell(\mathbf{y}_i, f(\mathbf{x}_i; \boldsymbol{\theta}_t)),$$

where $\boldsymbol{\theta}$ represents the model parameters $\mathbf{W}$ and $\boldsymbol{b}$ and $\boldsymbol{\theta}_t$ indicates the value of the parameter $\boldsymbol{\theta}$ at step t. So, by using the derivatives computed by the backpropagation algorithm, the update rule for each layer $l \in \{1, ..., L\}$ can be rewritten using following updates rules

$$\begin{cases} \boldsymbol{W}_{t+1}^{(l)} = \boldsymbol{W}_t^{(l)} - \gamma \nabla_{\boldsymbol{W}_t^{(l)}} \ell \\ \boldsymbol{b}_{t+1}^{(l)} = \boldsymbol{b}_t^{(l)} - \gamma \nabla_{\boldsymbol{b}_t^{(l)}} \ell \end{cases}.$$

2.2.4 Learning sequence

The difference between parameters and hyperparameters is first presented in Section 2.2.4.1. Section 2.2.4.2 presents **model optimisation** whose purpose is to select the best model to handle a given problem. Finally, Section 2.2.4.3 presents **model assessment**, which aims to evaluate the general performance of the selected model. This allows to compare several "best" models. The learning sequence refers to the period during which model optimisation and evaluation are carried out. In addition to the content of the courses [21] and [37], this section was written using [39], [42], [53] and [55].

2.2.4.1 Parameters and hyperparameters

According to the previous sections, a chosen model has several parameters, weights and biases, that are adjusted during training. However, they are not the only parameters involved in the learning process. Indeed, other parameters, called **hyperparameters**, are not modified by learning but control the learning process. They must be tuned manually.

Recall that a **sample** is an element of the dataset. For instance, in our case of ship detection, a sample corresponds to a single image. The main hyperparameters are the following ones.

- **Number of iterations:** one iteration is the period during which one batch of samples is seen by the model during the training process.
- **Number of epochs:** one epoch corresponds to the period during which the entire training dataset passes once through the model during the training step.

- **Batch size:** the entire training dataset (called batch) is divided into non-overlapping batches, which are called **mini-batches**. The samples of a mini-batch are used for one iteration of the training step and have been selected randomly or sequentially. This is a bit confusing, but the **batch size** refers to the size of a mini-batch, i.e. the number of training samples seen during one iteration.[14] This size affects training time and model quality. A small batch size implies fast training, but the estimations may not be accurate due to noise, while a large batch size implies a slower training time with more accurate estimations. Therefore, a common choice of batch size is to take a power of 2 between 16 and 512.[15]

- **Subdivision:** it is the number into which the mini-batch will be divided. The resulting sets can be called **"mini-mini-batches"**. For instance, if subdivision is equal to 4, the mini-batch is divided in 4 sets. If the batch size (size of the mini-batch) is 64, then the size of the resulting mini-mini-batches is equal to $64/4 = 16$. The samples are then loaded in sets of 16 instead of the full mini-batch at once and when the 4 mini-mini-batches have been loaded, the model parameters can be updated and the iteration is over. The subdivision parameter is useful for computers that do not have enough memory to store the information of a full mini-batch since it reduces the use of the GPU memory. Thus, by increasing this parameter, less internal memory is used, but training is slower.

- **Learning rate:** this parameter controls the step size made at each iteration while getting closer to a minimum of the loss function using the update rules (2.2), (2.3) or (2.4) seen in Section 2.2.2.

  If the learning rate is large, training is fast but a minimum of the loss function may be missed, leading to a sub-optimal solution. If it is small, a local minimum or even the global minimum can be reached, but with slower training since few changes are made between update rules. In extreme cases, if the learning rate is too small, it may get stuck in a poor local minimum, leading to a sub-optimal solution. If it is too large, divergence may occur, leading to a sub-optimal solution as well. The difficulty is therefore to set a good learning rate. The range of values for this parameter is between 0 and 1. In practice, a learning rate of 0.1 or 0.01 is a good starting point before tuning it.[16]

- **Momentum:** this hyperparameter can be added to the update rule in gradient descent. Noting this rule $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \gamma \boldsymbol{g}_t$, where γ is the learning rate and $\boldsymbol{g}_t$ indicates the gradient of the loss function, whose form varies according to the selected gradient descent method, it becomes

$$\begin{aligned} \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \boldsymbol{u}_t \\ \boldsymbol{u}_t &= \alpha \boldsymbol{u}_{t-1} - \gamma \boldsymbol{g}_t \end{aligned}$$

[14]If the batch size is equal to the size of the training set, then this corresponds to batch gradient descent. If it is taken equal to 1, this corresponds to stochastic gradient descent, see Section 2.2.2. In other words, in the first case, the entire training dataset is used during one iteration of the training step, while in the second case a single sample is used for one iteration.

[15]It is often suggested to use such values as a batch size to improve the efficiency of the training from a computational point of view. However, some references such as [7] and [44] state that taking values to the power of 2 does not necessarily improve performance while still being good starting values.

[16]A "good" learning rate depends on the problem and is not fixed. The learning rate can be set to one of these default values before finding the suitable one for a specific problem.

where α is the momentum and $\boldsymbol{u}_t$ is the velocity. The momentum controls the impact of recent gradients on the current update by giving them more or less weight. So, it applies a penalty when there are large weight changes in recent iterations and accelerates the update when there is almost no change. The advantage of using momentum is that it avoids getting stuck in a local minima and speeds up convergence. It ranges from 0 to 1, but is generally set to 0.9. It is also important to ensure that the learning rate is strictly smaller than the momentum term.

- **Decay:** when this hyperparameter is used, a term is added to the loss function L, which becomes
$$L(\boldsymbol{\theta}) + \delta||\boldsymbol{\theta}||^2,$$
where δ is the decay value and $\boldsymbol{\theta}$ indicates the model parameters (weight and bias). This prevents *overfitting* by not letting the model become too complex. The decay generally takes a value between 0 and 0.1. It should be set high enough to reduce the risk of overfitting, but not too high because the model might not be complex enough, which leads to a trade-off when selecting this hyperparameter value. In addition, small values seem more appropriate for large datasets and deep architectures.

- **Steps-Scales:** at the beginning of training, no information has been collected yet, so a high learning rate is required. However, during training, information is accumulated and such a high learning rate is no longer appropriate, making it worthwhile to modify it. This is the purpose of the step-scale hyperparameter, where step corresponds to a number of iterations at which the learning rate is multiplied by scale. It is also possible to have several steps with several associated scales. For example, if steps is fixed to 6000-6750 and scales to 0.5-0.2. This means that the value of the learning rate first remains constant until the 6000th iteration, at which it is multiplied by 0.5, then it remains constant again at this new learning rate value until the 6750th iteration, at which it is itself multiplied by 0.2. The learning rate value then remains unchanged until the end of training.

To summarise, for an epoch to be completed, the entire dataset must be passed through the network. However, the dataset is divided into m mini-batches of size b. Each mini-batch passes through the network during one iteration, so it takes m iterations for all the m mini-batches, i.e. the whole dataset, to pass through the network. Hence, there are m iterations in one epoch.

The number m of mini-batches is not the same as the batch size b. If the training dataset is of size n, then the following equality must be verified $n = mb$. This implies that there are $\frac{n}{b}$ iterations in one epoch. If n is not divisible by b and there is a remainder r, the simplest solution is to perform an additional iteration using these r samples.

This leads to one main relationship between these hyperparameters: the total number of iterations is $\frac{n}{b}e$, where e is the number of epochs.

2.2.4.2 Model optimisation

For a given type of network, model optimisation aims at selecting the best model by choosing the best hyperparameters. For this purpose, the provided dataset must be split in different parts. The splitting method depends on the size of the dataset and on the objectives.

Train - validation - test method

This method is suitable for large datasets.

1. Randomly split the dataset into 3 subsets, called **training**, **validation** and **test** datasets.
2. a) On the training dataset: fit the model by adjusting its parameters.
   b) On the validation dataset: get an evaluation of the resulting model using performance measures.
   c) Is the model good enough?
      ∗ No: tune the hyperparameters of the model in order to optimise the performance measures. Restart step 2. with these updated hyperparameters.
      ∗ Yes: fix the final parameters and hyperparameters of the model and then, go to step 3.
3. On the test dataset: get an evaluation of the final model using performance measures.

When the initial dataset is divided into three subsets, there must be enough observations in the training dataset, otherwise either overfitting[17] or underfitting[18] may occur due to this lack of data. The validation and test datasets should not be taken too small either, as in both cases the performance measures evaluated on these too small datasets will have a large variance. On the validation dataset, this would not lead to proper hyperparameter tuning, while the metrics evaluated on the test dataset would not represent the real performance of the model. A general choice is to split the observations as follows: 60% of the observations in the training dataset, 20% in the validation dataset and 20% in the test dataset. For instance, another option could be to take a splitting of 80/10/10.

K-fold cross-validation

This method is suitable for small datasets, large unbalanced datasets, i.e. datasets where the number of samples in each class is not approximately the same, or for more robust model optimisation in large datasets.

1. Randomly split the dataset into 2 subsets, called **training** and **test** datasets. For example, the splitting can be made using proportions of 80% and 20%.
2. Randomly split the training dataset into K folds.
3. a) For each fold :
      ∗ Consider the union of the other folds as the training dataset.
      ∗ On the training dataset: fit the model by adjusting its parameters.
      ∗ Use the selected fold as validation dataset.
      ∗ On the validation dataset: get an evaluation of the resulting model using performance measures.

[17]**Overfitting** occurs when the model fits too well to a specific dataset, e.g. the one on which it was trained, and therefore makes poor predictions on new data from a similar problem. This means that the model does not generalise well. In other words, the predictions made on the training dataset are very accurate but those made on the test dataset are not accurate at all.

[18]**Underfitting** occurs when the model does not fit the data sufficiently and is too simple. It is unable to capture the relationship present in the data. This results in poor predictions on the training and test datasets.

   b) Compute the average performance measures obtained on the K folds.[19]
   c) Is the model good enough?
      * No: tune the hyperparameters of the model in order to optimise the performance measures. Restart step 3. with these updated hyperparameters.
      * Yes: fix the final hyperparameters of the model and then, go to step 4.
4. On the complete training dataset (not split into K folds): train the model with the chosen hyperparameters.
5. On the test dataset: get an evaluation of the final model using performance measures.

This method aims to evaluate the ability of the model to train and to adapt to new data in a robust way. Actually, K-fold cross-validation only corresponds to steps 1., 2., 3.a) and 3.b). The complete process described above is the most commom way to perform model optimisation after doing K-fold cross-validation. Thus, this allows to perform model optimisation and model evaluation.

Nested cross-validation

Nested cross-validation is a more robust method to perform model optimisation and model evaluation based on the cross-validation principle. This corresponds to applying K-fold cross-validation for model selection and N-fold cross-validation to get a robust evaluation of the performance measures of the final model.

I. Randomly split the dataset into N sets.
II. **(Outer loop)** For each set n, with $n = 1, ..., N$,
   1. Consider set n as the test dataset.
   2. Consider the union of the other folds as the (outer) training dataset.
   3. Randomly split the (outer) training dataset into K sets.
   4. a) **(Inner loop)** For each fold k, with $k = 1, ..., K$,
         * Consider the union of the other folds as the (inner) training dataset.
         * On the (inner) training dataset: fit the model by adjusting its parameters.
         * Use the selected fold k as validation dataset.
         * On the validation dataset: get an evaluation of the resulting model using performance measures.
      b) Compute the average performance measures obtained on the K validation folds.
      c) Is the model good enough?
         * No: tune the hyperparameters of the model in order to optimise the performance measures. Restart step 4. with these updated hyperparameters.
         * Yes: fix the final hyperparameters of the model and then, go to step 5.
   5. On the (outer) training dataset: fit the model by adjusting its parameters.
   6. On the test dataset: get an evaluation of the resulting model using performance measures.
III. Compute the average performance measures obtained on the N test folds.

[19]The same model, which is chosen before starting the process, is trained K times independently, i.e. the parameters obtained for one training are not retained for the following trainings. Only the performance measures are retained.

Pre-trained models

In previous paragraphs, it was explained that model parameters are updated during training. Instead of starting the learning process from scratch (by randomly initialising the weights), which can take days or weeks, a common way to start it is to initialise the weights from a publicly available trained model that is called a **pre-trained model**. Such a model has been trained on a large reference dataset to solve a problem similar to our problem. It can be used as a feature extractor or for a smart initialisation of the parameters. In other words, some information has already been learned during training on the other dataset and, then, applying it to our dataset makes training faster. Using a pre-trained model is called **transfer learning**.

Loss function

The loss function was introduced in Section 2.2.2. It is now appropriate to discuss how this function is used during model optimisation. Recall that the loss function evaluates how well the model fits some data, i.e. how well it predicts the expected results. During training, the algorithm uses it to determine whether its predictions are good enough, and if not, it keeps searching for a set of parameters that provides a smaller value for the loss function. Thus, on the training dataset, this function is an indicator of the learning progress and should decrease over time. On the validation dataset, the loss function evaluates the adaptation of the model to new data. At first, the value of the loss function should decrease, but at some point, the model starts to fit the training set too much, leading to overfitting, which causes the value of the loss function to increase. The difficulty then lies in determining the point at which the model fits the data neither too well nor too poorly. This is illustrated in Figure 2.4.

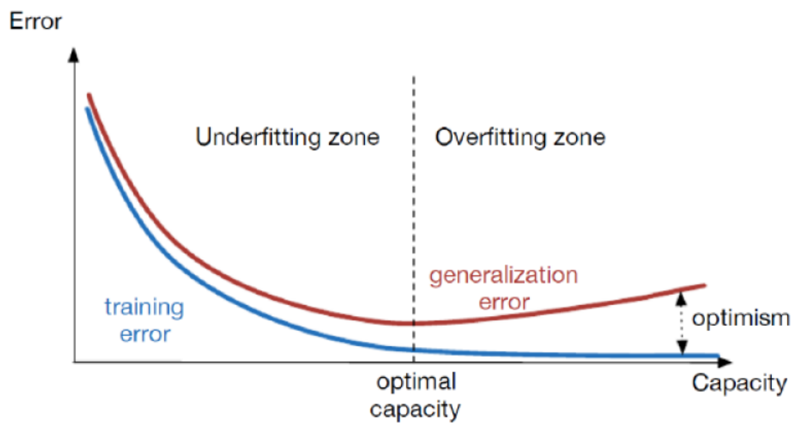


Figure 2.4: Evolution of the validation and training losses (here, training and generalization errors) over time and illustration of the optimal fitting between underfitting and overfitting. © [37]

Training can be stopped to prevent overfitting. This is called **early stopping**. The idea is to stop training as soon as the performance obtained on the validation set start to deteriorate. In that case, the obtained model parameters are stored before the last iteration.

2.2.4.3 Model assessment

This section presents **model assessment**. Once a model has been selected using model optimisation, it is possible to evaluate its performance on a new dataset, i.e. on the test dataset. The model assessment can either evaluate the best model selected or compare several best models. For this purpose, **performance measures** are computed on the test dataset. Those commonly used are presented in the rest of this section.

Performance indicators are metrics that evaluate the quality of the predictions made by a model on new data and provide an interpretable value of performance. One such metric is the **accuracy**. This is the main performance indicator and this has to be maximised. This corresponds to the number of correctly predicted outcomes over the total number of outcomes. This metric is defined as follows

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}, \tag{2.6}$$

where TP, TN, FP and FN stand for true positive, true negative, false positive and false negative, respectively. To understand the meaning of these values, the following table, the **confusion matrix**, is considered.

		PREDICTION	
		POSITIVE	NEGATIVE
ACTUAL	POSITIVE	True positive (TP)	False negative (FN)
	NEGATIVE	False positive (FP)	True negative (TN)

Example 2.2.1. To illustrate the idea behind these terms, we reuse the example of the classification of spam messages, seen in Section 2.1.1. In this case, a positive prediction corresponds to a message predicted as spam and a negative one corresponds to a message predicted as not spam.

- A **true positive** occurs when a model makes a positive prediction, which is correct. For instance, if the model predicts that a message is spam and that it is really a spam message.
- A **false positive** occurs when a model makes a positive prediction, which should have been classified as negative. For instance, if the model predicts that a message is spam, but this prediction is wrong because the message is actually not spam.
- A **true negative** occurs when a model makes a negative prediction, which is correct. For instance, if the model predicts that a message is not spam and that it is indeed not a spam message.
- A **false negative** occurs when a model makes a negative prediction, which should have been classified as positive. For instance, if the model predicts that a message is not spam, but this prediction is wrong because the message is actually spam.

A good model should maximise the number of true positives and true negatives, i.e. the number of correct detections, and should minimise the number of false positives and false negatives, i.e. the number of incorrect detections. The accuracy is one metric to compute this. Indeed, it is defined as the ratio of good detections over the total number of detections, see Formula (2.6).

The accuracy metric is problematic when the dataset is unbalanced. For instance, if there are a lot of non-spam messages and only a few spam messages, it is sufficient that the model always predicts messages as non-spam to achieve good accuracy. Indeed, it implies that only few detections are wrongly classified, these are spam messages classified as non-spam. To counter this problem and to penalise a false positive with a different weight than a false negative, new indicators that completely describe the performance are introduced:

- **Precision** is a measure of the number of positive predictions that are correct. In our example, out of all the messages that the model detects as spam, how many of those are correct. This metric is defined as follows
$$precision = \frac{TP}{TP + FP}.$$
- **Recall** is a measure of the number of truly positive elements that are correctly classified by the model. In our case, out of all the spam messages, how many of them are correctly detected by our model. This metric is defined as follows
$$recall = \frac{TP}{TP + FN}.$$

These metrics provide values between 0 and 1 and must both be maximised to get a good model. However, a model predicting a positive outcome only for instances that are clearly positive, has a good precision but a poor recall. Similarly, a model making only positive predictions has a good recall but a poor precision.

Therefore, a good model needs a combination of good precision and good recall. For this purpose, the **F_1-score** can be used. It is given by the harmonic mean of precision and recall as
$$F_1 = \frac{2}{\dfrac{1}{precision} + \dfrac{1}{recall}} = 2 * \frac{precision * recall}{precision + recall}.$$
An advantage of this metric is that extreme values of precision and recall are punished.

An alternative performance measure to the F_1-score allowing both precision and recall to be maximised is the **mean average precision (mAP)**. This measure is based on the **average precision (AP)**, which is itself based on the **precision-recall curve**.

- In general in machine learning, before assessing the quality of the model, a confidence threshold is often used to remove uncertain predictions. For example, if the threshold is set at 0.7, only those predictions for which the model is at least 70% sure that they are correctly predicted are conserved. Thus, the higher is the threshold, the higher the precision metric, but the lower is the recall. Hence, precision and recall have antagonistic behaviours according to the threshold. The curve obtained by reporting the precision-recall values for different confidence threshold values is called the **precision-recall curve**. It illustrates a trade-off for several threshold values between the precision and recall metrics.

- The AP value summarises the area under the precision-recall curve into a single scalar. A good AP value occurs when both precision and recall are high. The perfect value, i.e. equal to 1, means that all positive predictions are accurate and no negative predictions are classified as positive. Thus, maximising the AP value involves maximising precision and recall. Inversely, if both of these values are low, the AP value is also low. The advantage of the AP lies in its independence from the choice of a confidence threshold, which is sometimes difficult to set in practice.

Since AP can be computed for each class,[20] the mean of the average precisions over all classes can be computed. This provides the mAP value, which must be maximised. This performance measure is mainly used because it is simpler to maximise AP than to maximise all precisions and recalls. Furthermore, an advantage of the mAP over the F1-score is that it is independent of the choice of confidence threshold. This threshold may be difficult to set in practice, but it must be set before computing precision, recall and F1-score because their values vary according to the threshold. Thus, when the mAP is used, there is no need to set the confidence threshold and it indicates whether the model is stable and consistent based on several confidence thresholds. In practice, this measure is widely used for evaluating object detection models.

Remark 2.2.2. In object detection, another possibility is to visually analyse if the predictions are accurate or not, i.e. whether the detected objects correspond to the correct objects and are correctly located. Some performance measures for detection exist and are introduced in following sections, such as the intersection over union between the prediction and the ground truth.

2.2.5 Convolutional neural networks

A **convolutional neural network (CNN)** is a neural network which was firstly introduced to take visual data as input and whose original idea was based on the visual cortex of animals. As the name suggests, it is based on convolutions, which will be explained later in this section. In the context of ship detection, the purpose of such networks is to pre-process small amounts of information, for instance for feature extraction.

Before investigating the structure of this network, recall that in this master's thesis, greyscale images will be used in the practical part. Such an image corresponds to a set of pixels, each of them being represented by a real value. Thus, a greyscale image is represented by a 2-dimensional tensor of reals.[21] We note $\mathbf{X}$ the input of the network, where $\mathbf{X} \in \mathbb{R}^{p \times n}$ with p the height of the image and n its width. Coloured images are represented by three-dimensional tensors. A third dimension is needed to store the colour information since a coloured image is composed of the three bases: red, green and blue. Thus, the input of the network is $\mathbf{X} \in \mathbb{R}^{C \times p \times n}$, where C is the number of channels, which is equal to three in this case.

Figure 2.5 gives an example of the structure of a CNN. Here, the input image $\mathbf{X}$, a greyscale picture of a car, is fed to a CNN for feature extraction. This network is composed of several convolutional, pooling and fully connected layers, which will be introduced in this section. Based on the extracted features, the network makes a prediction, which corresponds to a car here.

In this example, rectangular parallelepipeds, referred to as "blocks" in the following for simplicity, are used to represent the changes in dimensions of an input when it is fed to a CNN and passes through several layers. Indeed, the layers change the shape of the input

[20]The AP can be computed for each class since it is based on the confusion matrix values, which are computed for a given ground truth class. In a classification problem, a class is a category.

[21]In deep learning, a tensor is considered as a generalised matrix as follows: a 0-dimensional tensor is a scalar, a 1D tensor a vector, a 2D tensor a matrix, a 3D tensor a vector of matrices of identical size, a 4D tensor a sequence of 3D tensors, etc. Thus, a greyscale image is represented by a 2D tensor and a coloured image by a 3D tensor. In the field of deep learning, tensors are widely used because they provide a representation of multidimensional data structures and because the associated tensor operations enable fast internal computations which are necessary in this field.

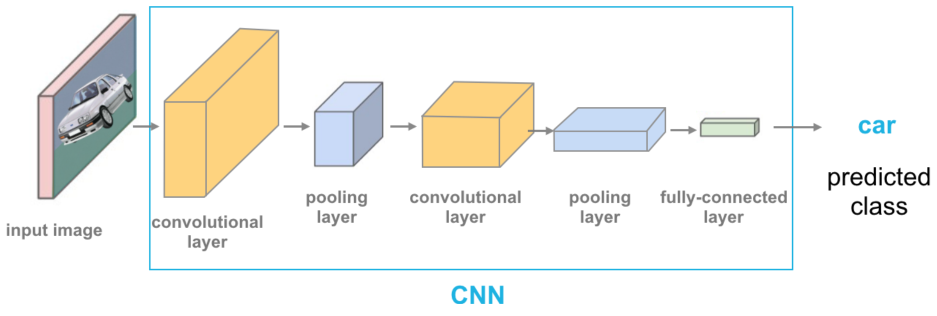


Figure 2.5: Example of a convolutional neural network structure. © [11]

image and of each following block. By modifying the dimensions, the CNN will capture the characteristics of the image better. The input image is represented by a rectangle, which is a flat block, since this corresponds to a 2-dimensional tensor. Following blocks are not flat and correspond to 3-dimensional tensors, a new dimension has been added compared to the input shape to extract image features. This processing was performed by a convolutional layer. The size of a block is reduced using a pooling layer. The last layer used in this architecture is a fully connected layer, which is used to flatten the 3-dimensional tensor produced by the previous layer into a vector. This vector summarises some information extracted by the CNN about the input image and is used to predict the input's category, a car in this case.

As mentioned above, a CNN is composed of convolutional, pooling and fully connected layers, which are presented in this section written using the materials of [35], [37] and [60]. The terms *depth*, *height* and *width* will be used to explain the changes made by a given layer on a block. For clarity, Figure 2.6 illustrates which term corresponds to which part of a block. For the input image $\mathbf{X} \in \mathbb{R}^{C \times p \times n}$, the block has a depth of length C, a height p and a width n. If this is a greyscale image, $\mathbf{X} \in \mathbb{R}^{p \times n}$ and the block has no depth.

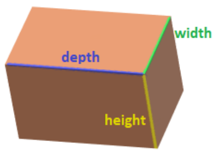


Figure 2.6: Illustration of what the terms *depth*, *height* and *width* refer to on a block.

2.2.5.1 Convolutional layer

The idea behind a convolutional layer is to apply the same linear transformation locally everywhere while preserving the signal structure. For this purpose, convolutions are used. For a simpler definition, the case of one-dimensional convolution, where one-dimensional tensors are used, is considered first.

Definition 2.2.1. For w, $W \in \mathbb{N}$ such that $w \leq W$, given an input vector $\mathbf{x} \in \mathbb{R}^W$ and a vector $\mathbf{u} \in \mathbb{R}^w$, called the *convolutional kernel*, the **discrete convolution $\mathbf{x} * \mathbf{u}$** is a vector of size $W - w + 1$ such that

$$(\mathbf{x} * \mathbf{u})[i] = \sum_{m=1}^{w} \mathbf{x}_{m+i-1}\, \mathbf{u}_m,$$

where $i \in \{1, ..., W - w + 1\}$.

Remark 2.2.3. The operation defined above actually corresponds to the *cross-correlation operation*. However, it is called convolution in the field of machine learning.

This can be generalised for multi-dimensional tensors. The focus will be on three-dimensional tensors, as this is the most common form in practice.

Definition 2.2.2. Considering

- h, H, w, W, $C \in \mathbb{N}$ such that $h \leq H$ and $w \leq W$,
- a 3D-tensor $\mathbf{X} \in \mathbb{R}^{C\times H\times W}$, called the *input feature map*, and
- a filter $\mathbf{U} \in \mathbb{R}^{C\times h\times w}$, [22] which slides across the input feature map, along its height and its width. The size of the filter determines the size of the receptive field, which is $h \times w$. The *receptive field* is defined as the region of the input feature map to which the layer has access and from which it extracts features.

The **convolution $\mathbf{X} * \mathbf{U}$** outputs a 2D-tensor $\mathbf{O}$ of size $(H-h+1)\times(W-w+1)$, called the *output feature map*. The tensor $\mathbf{O}$ is defined as

$$\mathbf{O}_{j,i} = \sum_{c=1}^{C}(\mathbf{X}_c * \mathbf{U}_c)[j,i] = \sum_{c=1}^{C}\sum_{n=1}^{h}\sum_{m=1}^{w}\mathbf{X}_{c,\,n+j-1,\,m+i-1}\;\mathbf{U}_{c,\,n,\,m},$$

where $j \in \{1,...,H-h+1\}$ and $i \in \{1,...,W-w+1\}$.

Actually, computing this means doing at each location the element-wise product between the filter and the input elements of the tensor that the filter overlaps and then, to sum up the results obtained for each location.

Figure 2.7 (a) illustrates the dimensions of an input, a kernel and the associated output produced by the convolution, which is a tensor of size $(H-h+1)\times(W-w+1)$ as explained previously. A figure illustrating how this convolution is computed, as well as examples of the definitions introduced above, is given in Appendix B.1.

Remark 2.2.4. It is also possible to apply D convolutions to produce an output tensor of size $D \times (H-h+1)\times(W-w+1)$, where D corresponds to the depth. In this case, something visually similar to Figure 2.7 (b) is obtained. For instance, if a 2D tensor is given as input to a convolutional layer, it is possible to produce a 3D tensor using multiple kernels. In particular, this can be applied to a greyscale input image, since it is represented by a 2-dimensional tensor.

Convolutional layers can take three additional parameters.

1. The **padding** parameter indicates the size of a frame of zeros added around the input tensor. This can be used to control the dimensions of the feature map. For instance, it is possible to keep these dimensions constant across all layers of the network. Indeed, without this parameter and using a filter whose width and height are larger than 1, the size of the layer's output will be smaller than its input. In other words, applying several convolutional layers causes the image to lose pixels at its edges. This problem can be solved using the padding parameter.

[22] The term "kernel" is used for a 2-dimensional structure and the term "filter" is mainly used for higher dimensions. The latter corresponds in fact to a collection of kernels.

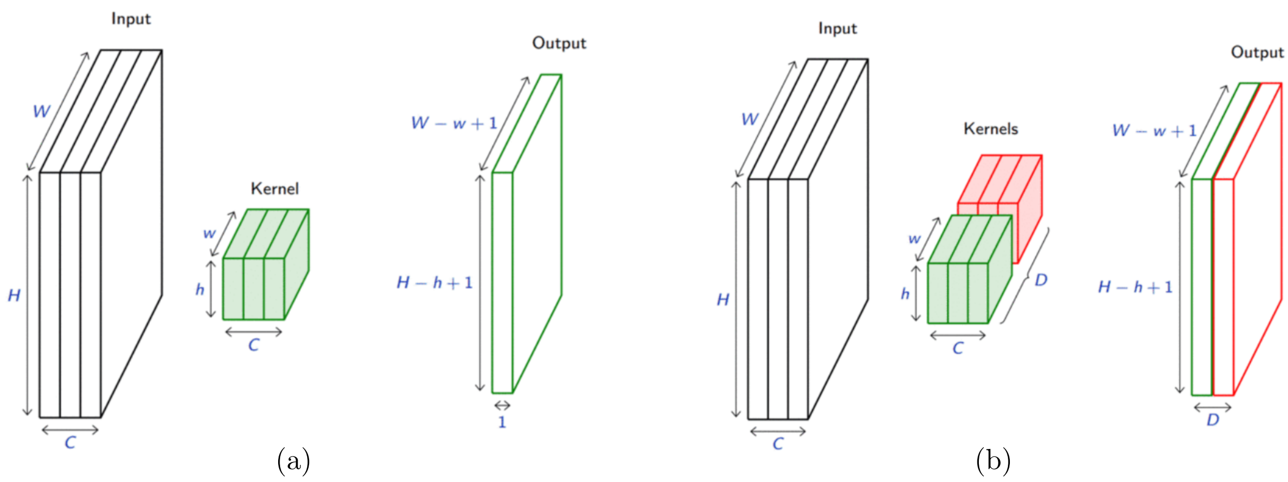


Figure 2.7: (a) Illustration of the dimensions of an input, a kernel and the output produced by the convolution. (b) Illustration of the application of D convolutions, where D is the depth. © [37]

2. The **stride** parameter indicates the step size while moving the kernel through the input tensor. This can be used to reduce the size of the feature map. This parameter is useful for computing purposes, as a large feature map is computationally expensive.
3. The **dilation** parameter modulates the expansion of the filter without adding weights. For this purpose, it adds rows and columns of zeros between values of the tensor. This parameter is useful if a large receptive field is required, as it allows the size of the field to be increased without modifying the number of parameters of the network. For this purpose, a dilation value larger than one must be taken. It is also possible to enlarge the size of the receptive field by increasing that of the filter, but this would increase the number of parameters and therefore the computational complexity.

To conclude about convolutional layers, these layers use filters which slide on the width and the height of images and produce new images with modified features. In practice, the filter will correspond to the weight matrix, which contains all parameters of the network.

The output of a convolutional layer can be linear. To eliminate the linearity, it is usual to apply an activation function just after this layer. This can also be useful to normalise the output. The ReLU activation function, presented previously and defined as

$$f(x) = \max(0, x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases},$$

is frequently used in CNNs because it also gets rid of negative values, which would be problematic when working with pixel values, i.e. with positive values.

Remark 2.2.5. The convolution function is said to be equivariant to any function that translates the input. This means that if an object moves in the input image, it will move with the same displacement in the output. This allows the position of an object to be preserved in an input image and in the output image obtained after the application of a convolutional layer.

2.2.5.2 Pooling layer

Pooling layers aim to reduce the size of the input while preserving its whole structure. This is done in a similar way to a down-scaling operation. This process is useful when the input volume is large.

Reducing the size of the feature map is not a problem when the previous layers of the network have already captured the important information. It is even preferable, as this reduction leads to smaller and more manageable representations, which also implies a decrease in the total computing time of the network training.

The idea of a pooling layer is to split the input feature map into several non-overlapping areas, called *pooling areas*, and then replace each of these areas by a single output value that summarises the feature present in that area. There exist several types of pooling layers, the main ones are presented below.

Definition 2.2.3. Consider a pooling area $h \times w$ and a 3D input tensor $\mathbf{X} \in \mathbb{R}^{C\times(rh)\times(sw)}$ with h, w, C, r and $s \in \mathbb{N}$. The **pooling** operation produces an output tensor $\mathbf{O} \in \mathbb{R}^{C\times r\times s}$.

- If the **maximum pooling** or **max-pooling** is used, the output tensor is given by

$$\mathbf{O}_{c,j,i} = \max_{n<h,\, m<w} \mathbf{X}_{c,\, r(j-1)+n,\, s(i-1)+m}$$

  where $c \in \{1, ..., C\}$, $j \in \{1, ..., r\}$ and $i \in \{1, ..., s\}$.

- If the **average pooling** is used, the output tensor is given by

$$\mathbf{O}_{c,j,i} = \frac{1}{hw} \sum_{n=1}^{h} \sum_{m=1}^{w} \mathbf{X}_{c,\, r(j-1)+n,\, s(i-1)+m}$$

  where $c \in \{1, ..., C\}$, $j \in \{1, ..., r\}$ and $i \in \{1, ..., s\}$.

Figure 2.8 illustrates the dimensions of an input and the output produced by applying a $h \times w$ pooling operation.

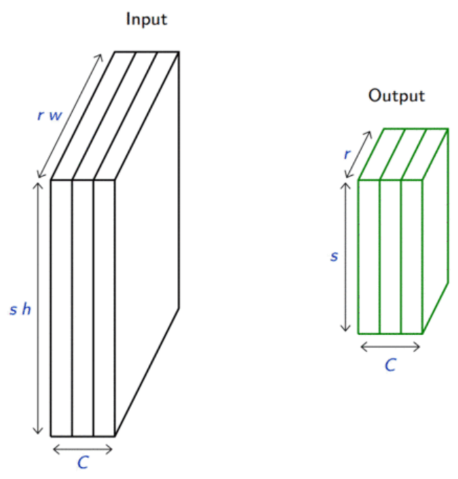


Figure 2.8: Illustration of the dimensions of the input and the output produced by applying a $h \times w$ pooling operation. © [37]

In Appendix B.2, there are some examples of the definitions of maximum and average pooling. There is also a figure illustrating how the output of a pooling operation is computed, i.e. how the output of Figure 2.8 is obtained.

The following example illustrates a real life case. As done previously, the outputs obtained at each layer of the network are illustrated with blocks since these are 3D-tensors.

Example 2.2.2. Figure 2.9 shows that the application of the pooling layer reduces the size of the block and of the image used as illustration. When a max-pooling layer has been applied, the scale of the image has been reduced by removing the least relevant information and retaining the main features. Conversely, when an average pooling layer has been used, more information is preserved, but the main features may be diluted.

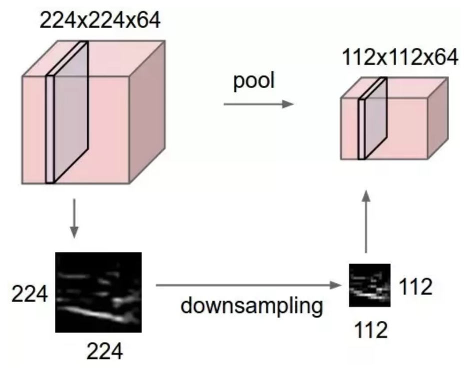


Figure 2.9: Illustration of a 2×2 pooling operation applied to a real image of size 224×224, providing an image of size 112×112. © [35]

Remark 2.2.6. Pooling layers are invariant to local translations as long as they are multiple of the pooling size h and w. This means that if there is an object in an image, after applying a pooling layer, the object will still be exactly the same and the information about it will not have changed, but the object may have moved slightly in the image. For example, the object might have been shifted to the right side of the image instead of being centered. This implies that pooling layers are useful to preserve the presence of a specific pattern whose precise location does not matter.

2.2.5.3 Fully connected layer

The last kind of layer used in a CNN is the **fully connected layer**. They were briefly explained in Section 2.2.1. They connect every node of a given layer to every node of the next layer. Having so many links is computationally expensive as it involves a large number of operations to perform. However, they are useful in the case of a CNN because they transfer all the information from the previous layer to the next one.

At the last convolutional or pooling layer, the output tensor is three-dimensional, but most of the time CNNs have to return vectors to be able to draw conclusions. To this end, the output of this last layer, which is a 3D tensor, is flattened into a vector through a flattening step. The flattened output is then fed to a fully connected layer that will compress the information again, so that the final output vector is produced. Figure 2.10 illustrates the architecture of a CNN, in which a flattening step is illustrated.

Remark 2.2.7. In this section, we have seen that the pooling and convolutional layers produce outputs with smaller size than the inputs.[23] In other words, this allows to reduce the number of pixels in the feature map to make it more manageable, i.e. to reduce the size of the feature map. In the following, the convolutional and pooling layers will therefore be used to perform this operation, called **downsampling**.

[23]For this, see Definitions 2.2.2, 2.2.3 and Remark 2.2.4.

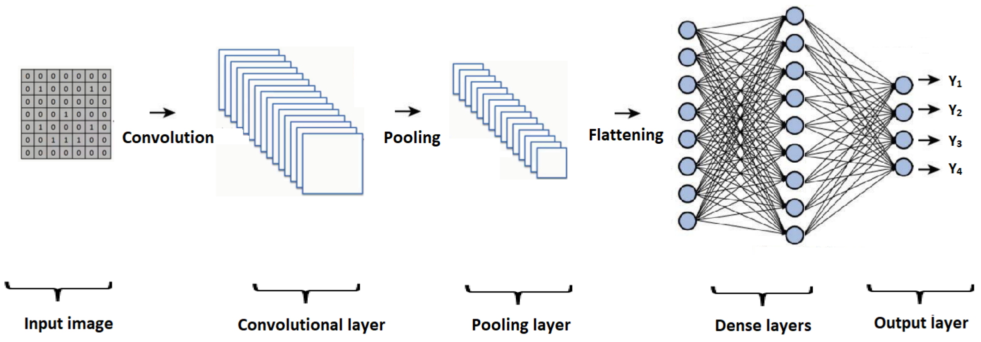


Figure 2.10: Architecture of a CNN with several fully connected layers (called here **dense layers**) at the end. © [38]

Remark 2.2.8. In some more complex architectures, **skip connections** are used. In a neural network, these connections allow to skip some layers of the network and to give as input to a further layer, an output value coming from the layer from which the skip connection starts. Without going into detail, these connections were originally introduced to solve different types of problems in some architectures. One possible use, for example, is for regularisation to reduce overfitting and improve generalization error. Nowadays, it is widely used, notably in the ResNet architecture [29], which will be discussed later. An example of this is shown in Figure 2.11. Only a part of the network is used to illustrate the difference between using or not using a skip connection.

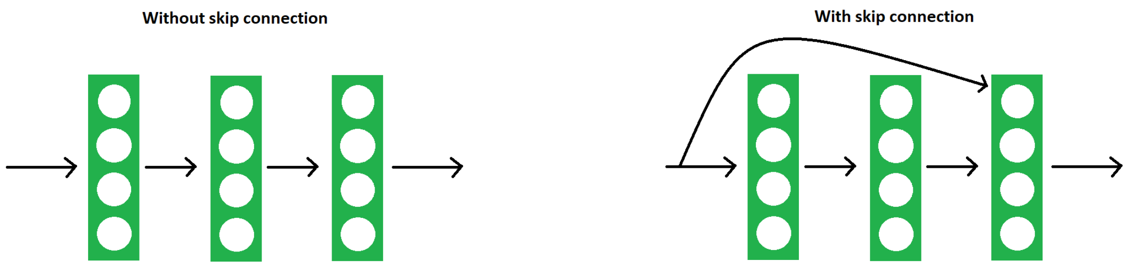


Figure 2.11: Illustration of a part of a network without and with a skip connection respectively.

2.3 Computer vision and object detection

This section aims to explain what **computer vision** is. It was written using the material of the *"Deep learning"* course [37]. Computer vision is a part of artificial intelligence, which uses many deep learning algorithms to focus on the interpretation of visual data (photos or videos). These data are represented by a set of pixels, which are all individually represented by a real value and which can not be interpreted as such by a human being. Hence, computer vision aims to extract information from these real numbers. For this purpose, this field deals with how programs can capture information and how to configure them for this purpose. In other words, the idea of this field is to make algorithms able to understand directly what is on some visual data, just as humans do when they think and analyse immediately what they see.

In computer vision, several tasks can be performed, each requiring a different neural network architecture. The following focuses a bit on classification, object localisation and semantic segmentation, and mainly on object detection. In computer vision, **classification** consists of identifying which category an image belongs to among a set of predetermined categories. This can be done using convolutional neural networks, which achieve state-of-the-art results for this kind of data. The goal of **object localisation** consists in predicting the object and its boundaries in a given image. For example, on the left of Figure 2.12, the group of sheep has been categorised as sheep and has been located since a bounding box indicates where the sheep are. This is object localisation. If the image is just classified as sheep and that no bounding box is provided, then this is just classification.

The goal of **object detection** is to detect instances of particular objects on a numerical image or video. This is a bit different from object localisation since the latter finds the most visible object in the image, whereas object detection detects all instances of the object. Both object localisation and object detection provide the boundaries associated with the located object. For example, on the right of Figure 2.12, the goal was to detect the three sheep individually.

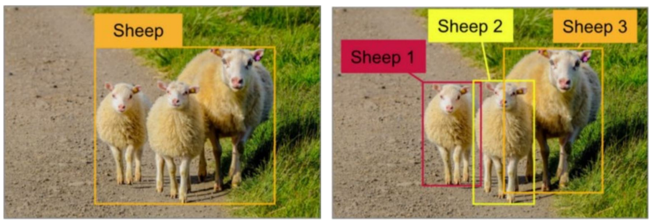


Figure 2.12: Object localisation (i.e. classification + localisation) (left) and object detection (right). © [37]

Semantic segmentation, or **image segmentation**, consists of clustering together different parts of an image representing the same object. In other words, each pixel in the image must be classified into one group. **Instance segmentation** has the same goal, but also detects if there are several instances of the same object class. For example, the image on the left in Figure 2.13 corresponds to semantic segmentation. Indeed, the image has been categorised into three groups, one for the grass, one for the road and one for the sheep. The image on the right illustrates instance segmentation since there are two clusters for the grass and the road, but there are also three additional clusters for each of the three sheep.

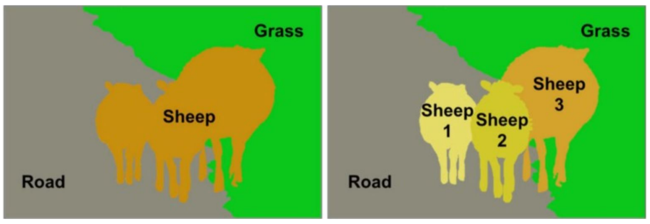


Figure 2.13: Semantic segmentation (left) and instance segmentation (right). © [37]

In the following, the concept of **data augmentation** will be used several times. It is important to keep in mind that the biggest limit to the performance of neural networks is the lack of data. Since collecting more data is usually expensive and difficult, and synthesising data is complicated and not necessarily representative of the true distribution, one solution consists in augmenting the data by applying base transformations on the data. This is a simple and efficient technique. Figure 2.14 illustrates some transformations, such as rotations or colour modifications, that can be applied to images in order to obtain new ones and extend the dataset. By doing this, the model will be more accurate and fewer errors will be made. Moreover, this can be used to reduce overfitting.

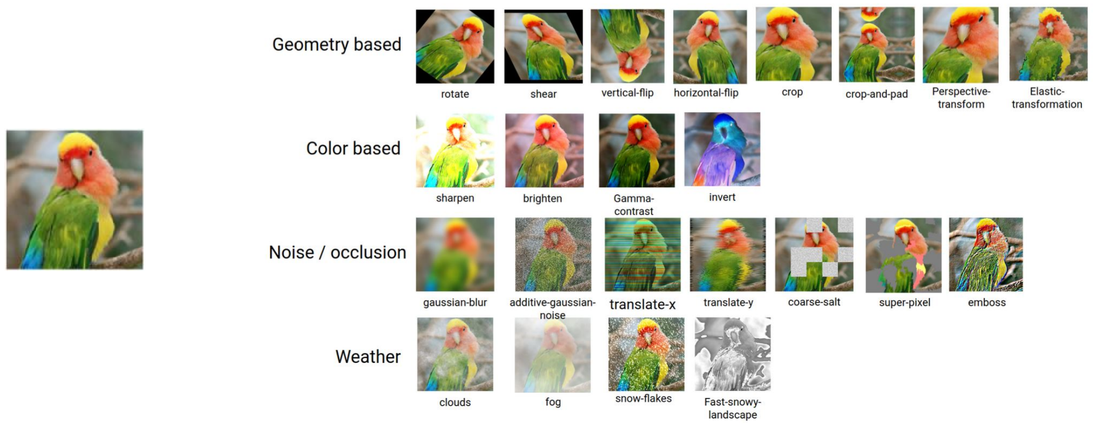


Figure 2.14: Data augmentation through base transformations on an image of a bird, the original image is on the left. © [37]

2.4 Some machine learning methods

This section presents machine learning methods that will be tested later on for ship detection. Some other methods, such as OverFeat, are also introduced to facilitate the description of the techniques of interest, or because they are close to them.

2.4.1 OverFeat

OverFeat [52] is an object detection algorithm consisting of two main parts. The first one is a convolutional neural network whose objective is to extract features from data. The second one is itself divided into two branches. The **classification head** classifies the object and outputs the confidence score related to this prediction. The **regression head** provides its bounding box. The two branches are trained independently. OverFeat's architecture is illustrated in Figure 2.15. This section is based on the content of [37] and [52].

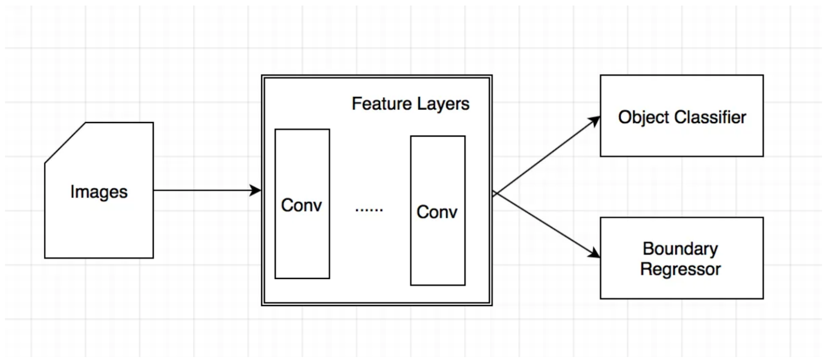


Figure 2.15: OverFeat architecture. © [20]

For both heads, the image on which the object detection has to be applied is resized at several scales. The advantage of multi-scale classification and localisation is that the model is more robust and precise.

The classification head

For each scale, classification is performed on several areas of the image. This is the concept of the sliding window: for an image of fixed size, a window (of smaller size) will slide on its height and its width. For each position of the sliding window, a classification is performed to detect if an object is inside. Thus, for each location and scale, the classifier head outputs a class and a confidence score, i.e. the class of the found object and the probability that it is really located there, respectively.

Figure 2.16 illustrates this sliding window on an image of a bear, which has been adjusted to different scales. Here, the object to be located is a bear. The window is yellow if the probability that an object is located in it is high, purple if the probability is low and blue if it is mitigated on the presence or not of an object. A black and yellow rectangle, summarising the confidence, is associated with each of these images and is displayed below them. In such a rectangle, there are as many pixels as there are windows in the image above. Thus, each pixel represents the confidence score associated with one of the windows. Yellow pixels indicate that the algorithm is confident that the object is located there on the image. On the contrary, black pixels indicate that the algorithm is not confident at all on the presence of the object. In doing so, the resulting yellow shape stands out from the rest of the rectangle, which is black, and therefore indicates the location of the object, a bear in this case.

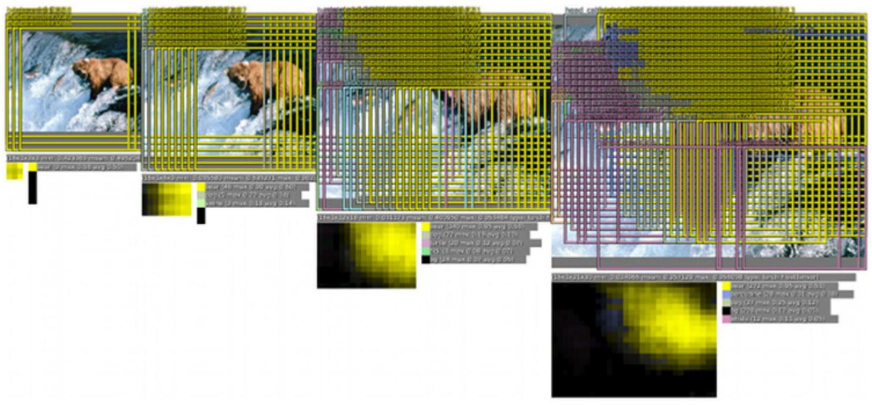


Figure 2.16: The classification head. © [52]

This sliding window does not slide at all possible positions. In other words, it does not shift by one pixel at a time. Indeed, this would be too computationally expensive. Here, the sliding window just slides on a coarse grid and does not test every position. In other words, it does not truly slide, as it takes fixed steps by testing precise predefined positions. Thus, the object is not delimited very precisely.

The regression head

The second branch uses a regression network to predict the coordinates of the object's bounding box, i.e. its precise position within the window. The first branch does not provide the coordinates of the box, it simply finds a coarse approximation of the object's location. Thus, the regression head produces 4 values for each window, corresponding to the coordinates of the bounding box, which should delimit the object more precisely.

By doing so, for each scale considered, several bounding boxes are predicted and are overlapping each other. This is illustrated in Figure 2.17. For each object and scale, there are too many predictions and some of them are equivalent. As with the classification head, a yellow and black rectangle indicates the confidence for the different bounding boxes.

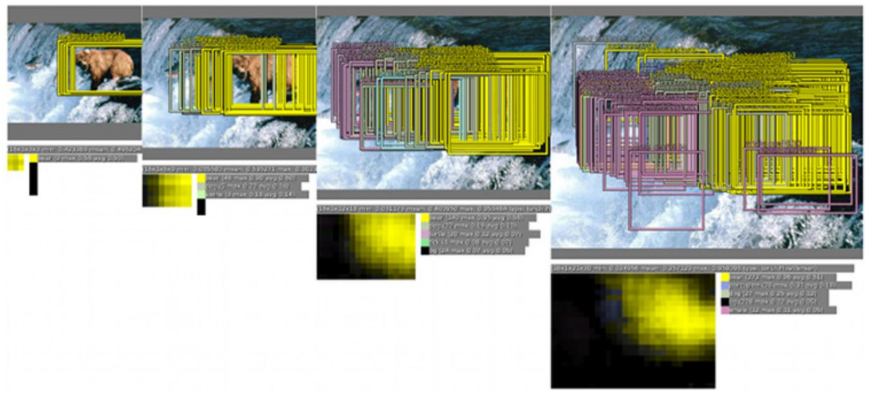


Figure 2.17: The regression head. © [52]

Thus, the final prediction, which should be the most suitable bounding box, is produced either by selecting the bounding box with the highest confidence score or by merging several bounding boxes together. This can be performed using a greedy merge strategy, whose corresponding algorithm is described in [52]. Figure 2.18 illustrates the final predicted bounding box, which includes the bear, as expected.

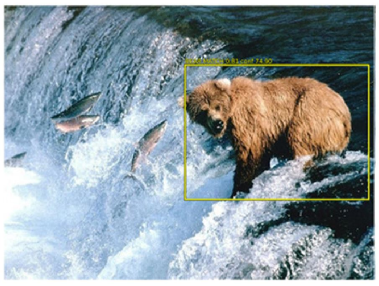


Figure 2.18: Object detection using OverFeat. © [52]

The neural network is then fully trained as follows: first, the network consisting of the feature extractor and classifier layers is trained. Next, the classifier layers are replaced by the regressor layers (and not the feature extractor layers, since they are common to both branches) and the network is then trained for localisation. The two branches are trained using their respective loss functions.

OverFeat has several drawbacks: there are two disjoint branches with their respective losses, which are not linked together; for an object detection problem, it should optimise detection and not location; and it can not reason about global context, which could result in inconsistent detections. The latter is due to the fact that OverFeat does not see the whole image during training, but rather sub-images (based on the positions of the sliding window), so it does not capture contextual information about the objects and their appearance, which means that it can not reason about the general content of the image.

2.4.2 YOLO

Introduction

Another algorithm for object detection is YOLO [46], "You Only Look Once", which models object detection as a regression problem. As before, the objective is to determine which objects are in the provided image and where they are. So, the purpose is to classify and locate the objects.

The main advantage of this algorithm is that, at test time, a single pass through the network is sufficient to get all bounding boxes' coordinates and class probabilities on the provided image. In other words, each image used as test set is given only once to the network, which was not the case for OverFeat, for it to locate and classify objects. YOLO can therefore detect objects in real-time, which makes it quite fast in comparison to other object detection algorithms.

As detection is performed by a single network, it can optimise detection, which was not the case for OverFeat that optimises location rather than detection. Another advantage compared to OverFeat is its ability to reason about global content when making predictions. Indeed, the whole image is seen during training. This implies the encoding of contextual information about classes and their appearance.

First, we present the first version of YOLO, based on [37] and [46], and then the various versions in which some problems have been solved. In particular, in Part II, we will focus on versions 3 and 4.

Principle of unified detection

YOLO is referred to as unified detection because unlike other object detection algorithms, the authors have unified the usual different parts present in a detection algorithm into a single algorithm, a single neural network.

YOLO is made up of three distinct parts. The first part consists in dividing the original image into a $S \times S$ grid. This is illustrated in Figure 2.19 with S equal to 7. In this case, the image is then divided into 49 cells.

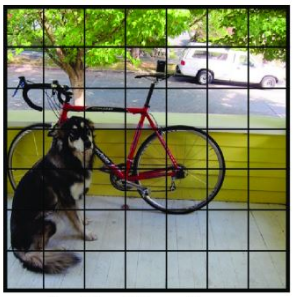


Figure 2.19: $S \times S$ grid on input image. © [46]

Each grid cell is responsible for detecting whether or not one object appears in it. A cell can detect a part of the object and not necessarily the whole object. Indeed, if the centre of an object appears into a grid cell, then that cell must detect it.

The second part is the bounding box regression.[24] Each grid cell predicts B bounding boxes. For each bounding box, 5 values are computed: 4 coordinates to describe its position

[24]This part is used only once when predicting the coordinates of the objects and their classes.

and a confidence score. This score represents the model's confidence that an object (or part of it) is in the associated bounding box. It also reflects the model's confidence in the accuracy with which it predicts this box.

For each grid cell, the probability that an object of a given class is located in that cell is computed. This score is the *class probability* and is calculated for each class. Thus, the highest score provides the class predicted for the cell. In other words, it is the most probable class for a given cell.

Thus, each grid cell predicts a vector of size $5B + C$, where 5 indicates the 5 values associated with each bounding box and C is the number of classes. So, the dimension of the output vector is $S \times S \times (5B + C)$.

In the previous example, we had $S = 7$. Now, if we take $B = 2$ bounding boxes and $C = 20$ classes of objects, the network will predict a vector of size $5B+C = 5\times2+20 = 30$ for each cell. Depending on the choice of these values, the output vector can be very large. In this case, there are $7 \times 7 \times 30 = 1470$ values to predict. The values taken for B and C are the ones used in the article [46] presenting YOLO. The setting of B equal to 2 is an arbitrary choice of the authors but that of taking C equal to 20 comes from the fact that they trained the model on a dataset containing 20 classes, i.e. the Pascal VOC 2012 dataset.

The images in Figure 2.20 illustrate the bounding boxes that can be obtained from the grid defined previously and the class probability map. This map is based on the class probabilities of each grid cell and assigns a colour to each of them, where each colour represents a class. So, the colour of a cell indicates the class of the object that is the most probable to be in that cell.

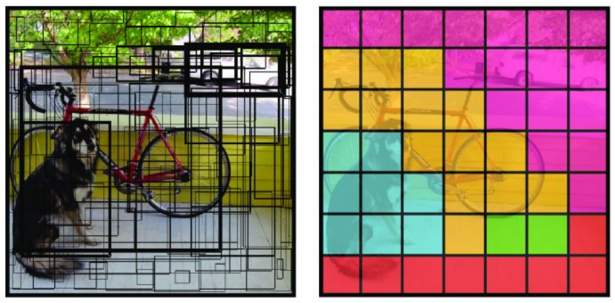


Figure 2.20: On the left image: bounding boxes and their confidence score, where the higher the confidence score is, the thicker the boundaries of the box are. On the right image: the class probability map, where the blue cells indicate that the class "dog" is the most probable one, the yellow cells correspond to the class "bicycle", the pink cells to the class "car" and the red cells to the class "dinning table". © [46]

The last part aims to select bounding boxes that perfectly fit the objects in the given image. Indeed, some of the bounding boxes found in second part do not bound correctly the object, they are either too large or they cut a part of the object, which makes them useless. The predicted bounding box should be the same as the *ground truth bounding box*, as illustrated on Figure 2.21. To this end, the "intersection over union" technique can be used which ensures that real boxes of objects and the predicted ones are equal. These ground truth boxes correspond to the annotated bounding boxes, which are available during training.

Intersection over union (IoU) is an indicator of performance for object detection. It evaluates the area of the intersection between a predicted bounding box $\hat{B}$ and the annotated bounding box B, over the area of their union. This is defined formally as

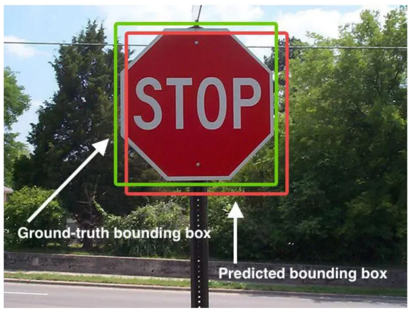


Figure 2.21: Illustration of the ground truth bounding box and a predicted bounding box. In this case, the prediction is not the most fitting bounding box of the object. © [50]

follows

$$\mathrm{IoU}(B, \hat{B}) = \frac{\mathrm{area}(B \cap \hat{B})}{\mathrm{area}(B \cup \hat{B})}.$$

The areas that are part of the formula are illustrated in Figure 2.22.

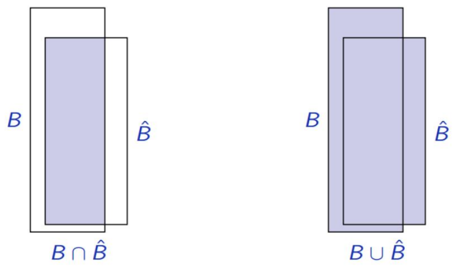


Figure 2.22: Illustration of area $(B \cap \hat{B})$ (left) and area $(B \cup \hat{B})$ (right). © [37]

If the ratio is equal to 1, the two bounding boxes are the same. If it is close to one, they are very similar to each other. Thus, during training, IoUs are defined based on the annotated bounding boxes which allows to eliminate predicted boxes that are not similar enough to the ground truth bounding box. Indeed, by using non-maximum suppression based on IoU,[25] YOLO provides unique bounding boxes that fit the corresponding object as well as possible. Figure 2.23 illustrates the value of the IoU score according to the positions of the boxes.

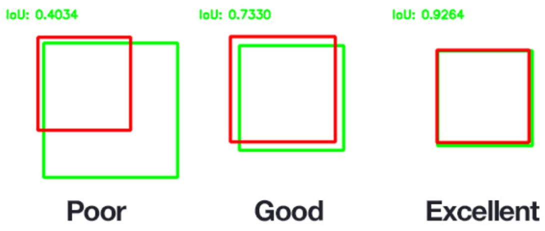


Figure 2.23: Illustration of the value of the Intersection over Union score according to the positions of the bounding boxes, the red one being the ground truth bounding box and the green the predicted bounding box. © [50]

[25]The principle of the non-maximum suppression (NMS) algorithm is the following: when several boxes have an IoU with the ground truth bounding box greater than a given threshold, only the box with the highest confidence score is retained.

This is the situation where ground truth bounding boxes are available. However, during inference, they are not available and the non-maximum suppression can not select the best prediction based on the ground truth. In this situation, the IoU used in non-maximum suppression still aims to remove duplicate bounding boxes associated with the same object, but its use is slightly different. Instead of computing the IoU between the prediction and the ground truth, the IoU is computed between the predictions of the same object. If the resulting IoU is high, the boxes seem to correspond to the same object and the prediction that is kept is the one with the highest confidence score. The other is rejected. If the resulting IoU is low, this means that the predicted bounding boxes are probably not associated with the same object. The decision whether to keep these boxes or not is then made individually on the basis of their confidence score.

Back to the example used earlier, by applying this technique, YOLO finally provides three bounding boxes, one for the dog, one for the bike, and one for the car, see Figure 2.24.

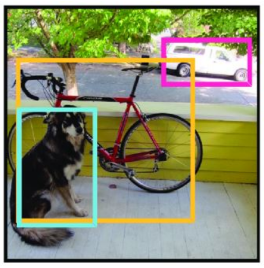


Figure 2.24: Final detections of a dog (blue), bike (yellow) and car (pink). © [46]

Figure 2.25 shows how each part relates to each other on our example.

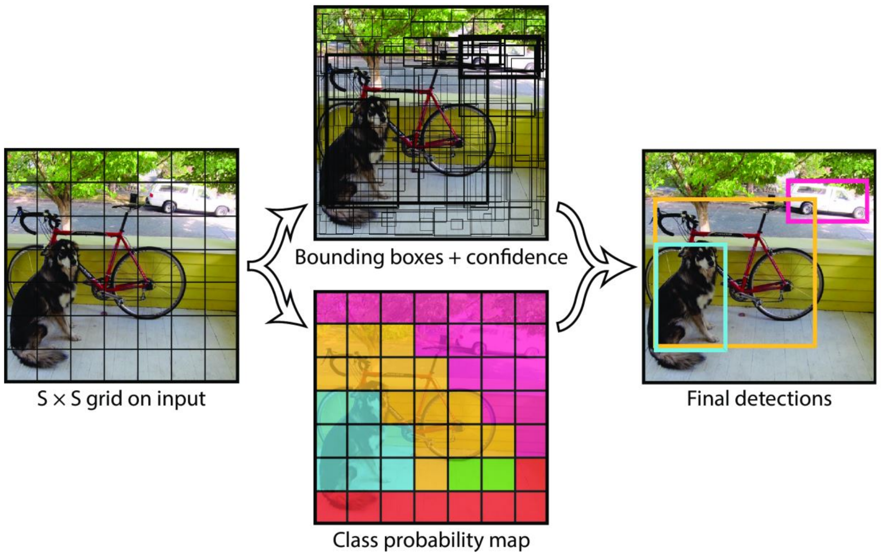


Figure 2.25: YOLO principle. © [46]

Network design

In this section, we still focus on the first version of YOLO, whose model is implemented as a convolutional neural network. It is composed of 24 convolutional layers, whose goal is to extract features of the image, followed by 2 fully connected layers, which predict the output probabilities and the coordinates of the bounding boxes. Between the convolutional layers, there are a total of four 2×2 pooling layers which reduce the size of the output feature maps used as input for the following convolutional layers. In the network, there are many 1×1 and 3×3 convolutional layers which alternate to decrease the dimension of the features space from previous layers.

The final output of the YOLO model is a $S \times S \times (5B + C)$ vector of predictions. By setting $S = 7$, $B = 2$ and $C = 20$, this final output is a $7 \times 7 \times 30$ tensor. The YOLO architecture is represented on Figure 2.26.

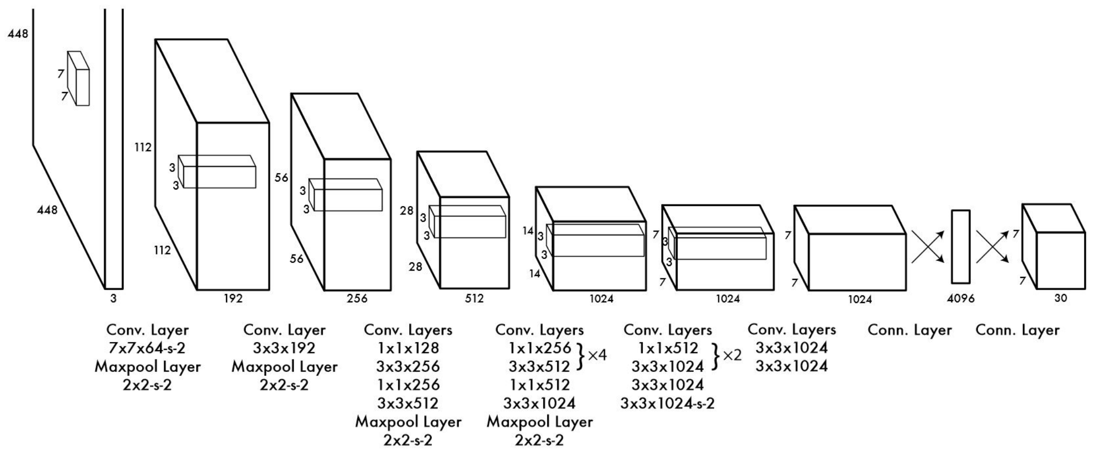

448
7
7
448
3
112
3
3
112
192
56
3
3
56
256
28
3
3
28
512
14
3
3
14
1024
7
3
3
7
1024
7
7
1024
4096
7
7
30
Conv. Layer 7x7x64-s-2 Maxpool Layer 2x2-s-2
Conv. Layer 3x3x192 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x128 3x3x256 1x1x256 3x3x512 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x256 3x3x512 ×4 1x1x512 3x3x1024 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x512 3x3x1024 ×2 3x3x1024 3x3x1024-s-2
Conv. Layers 3x3x1024 3x3x1024
Conn. Layer
Conn. Layer

Figure 2.26: YOLO architecture, where "Conv. layer" stands for a convolutional layer, "Maxpool layer" for a maximum pooling layer, "Conn. layer" for a fully connected layer and "s-2" indicates that the stride parameter is taken equal to 2. As explained in the text, there are a total of 24 convolutional, 4 maximum pooling and 2 fully connected layers. © [46]

Remark 2.4.1. The authors of YOLO also introduced "Fast YOLO" [46], which is a fast version of YOLO. This version contains only 9 convolutional layers instead of 24 and has fewer filters in those layers.

Training

At this point, the YOLO principle and architecture have been presented. The following aims to describe its training. As explained in Section 2.1.1, given a model and a dataset associated to a particular task, training consists in fitting the model by adjusting the model parameters to the provided data. Here, the task is to detect objects on images. Thus, the dataset must be composed of images, contain classes of objects present in these images, and indicate the locations of objects present in these images, i.e. the ground truth bounding boxes. Once the model is trained, the model parameters have been adjusted to the provided data and new data can be given to this model, which will be able to detect objects on these new images.

To reduce overfitting, the authors who introduced YOLO had applied data augmentation to their dataset before training it, see Section 2.3. Random scaling, translation and colour modifications, i.e. modifications of the exposure and saturation of the image, have been applied. There exist also other changes. For instance, flipping the data allows to distinguish the left and right side of objects or re-scaling allows to detect the same object of different sizes, in case it occupies different parts of the image.

YOLO uses transfer learning to make training quicker, as explained in Section 2.2.4.2. Indeed, the authors have pre-trained the first 20 convolutional layers of the YOLO architecture followed by an average pooling layer and a fully connected layer on the ImageNet dataset [31] (an image database) for a classification task. This network has been trained for approximately one week and the resulting pre-trained network is publicly available.

Once this pre-trained model is available and able to classify images, it must be converted to perform detection rather than classification. To improve performance, the last two layers of the pre-trained network, i.e. the average pooling layer and the fully connected layer, have been replaced by 4 convolutional layers and 2 fully connected layers. These additional layers are initialised with random weights, unlike the previous layers which are initialised with the weights of the pre-trained model. The last layer predicts class probabilities and bounding boxes' coordinates.

The following introduces the YOLO loss. Recall that this function allows to evaluate how well the model predicts the expected outputs and that the closer its value is to 0, the better the predictions made by the model are. During training, any of the $S \times S$ cells is assumed to contain at most (the center of) a single object. Given an image, we define cell index $i = 1, \ldots, S \times S$, predicted box index $j = 1, \ldots, B$ and class index $c = 1, \ldots, C$. Now, we introduce some notations.

- The following value indicates the presence of an object in a given cell.

$$\mathbb{I}_i^{\text{obj}} = \begin{cases} 1 & \text{if there is an object in cell } i \\ 0 & \text{otherwise} \end{cases}$$

- For each cell, several bounding boxes are predicted. The following binary value indicates which box is the most fitting one, i.e. the one with the value closest to 1 for IoU.

$$\mathbb{I}_{i,j}^{\text{obj}} = \begin{cases} 1 & \text{if there is an object in cell } i \text{ and predicted box } j \text{ is the most fitting one} \\ 0 & \text{otherwise} \end{cases}$$

- The following value indicates the presence of an object of a particular class in a given cell.

$$p_{i,c} = \begin{cases} 1 & \text{if there is an object of class } c \text{ in cell } i \\ 0 & \text{otherwise} \end{cases}$$

- To describe the bounding box, the values x_i, y_i, w_i and h_i are used. The couple (x_i, y_i) are the coordinates of the center of the box, w_i is the width and h_i is the height of the box. These values are defined only if $\mathbb{I}_i^{\text{obj}} = 1$. The bounding box is relative in location and scale to the cell i.

- The IoU between the predicted box and the ground truth target is noted as $c_{i,j}$.

In addition, the notations $\hat{p}_{i,c}, \hat{x}_{i,j}, \hat{y}_{i,j}, \hat{w}_{i,j}, \hat{h}_{i,j}$, and $\hat{c}_{i,j}$ are also used. They refer to the estimations provided by the network for each of the B bounding boxes (index j) per grid cell (index i).

First, the values $p_{i,c}, x_i, y_i, w_i, h_i$, and $c_{i,j}$ are computed from the ground truth data and the value of $\mathbb{I}_{i,j}^{\text{obj}}$'s and $c_{i,j}$ are computed for each cell index i and predicted box j. Then, the training procedure aims to minimise the following loss. This is mainly based on a sum-squared error because this is easy to optimise and it can maximise average precision.

$$\lambda_{\text{coord}} \sum_{i=1}^{S\times S} \sum_{j=1}^{B} \mathbb{I}_{i,j}^{\text{obj}} \Big((x_i - \hat{x}_{i,j})^2 + (y_i - \hat{y}_{i,j})^2 + (\sqrt{w_i} - \sqrt{\hat{w}_{i,j}})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_{i,j}})^2 \Big) \tag{2.7}$$

$$+ \lambda_{\text{obj}} \sum_{i=1}^{S\times S} \sum_{j=1}^{B} \mathbb{I}_{i,j}^{\text{obj}} (c_{i,j} - \hat{c}_{i,j})^2 + \lambda_{\text{noobj}} \sum_{i=1}^{S\times S} \sum_{j=1}^{B} (1 - \mathbb{I}_{i,j}^{\text{obj}}) \hat{c}_{i,j}^2 \tag{2.8}$$

$$+ \lambda_{\text{classes}} \sum_{i=1}^{S\times S} \mathbb{I}_{i}^{\text{obj}} \sum_{c=1}^{C} (p_{i,c} - \hat{p}_{i,c})^2 \tag{2.9}$$

1. **Coordinate loss:** the first term (2.7) of this loss computes the square error between the coordinates of the true bounding box and the predicted one, this checks whether the prediction covers the object. This is computed only if $\mathbb{I}_{i,j}^{\text{obj}} = 1$, i.e. if an object is in cell i and the predicted box j is the most fitting one for this cell. For the width and the height, the squared error has been computed on the squared root value.[26]

2. **Objectness loss:** the first part of the second term (2.8) penalises the prediction score of bounding boxes responsible of predicting objects. It evaluates the quality of the IoU produced by the network compared to the expected one, when considering cells which contain an object. As YOLO predicts several bounding boxes per grid cell and as there should be only one bounding box per object, the one selected is the one that has the highest IoU score with the ground truth. The second part of (2.8) aims to keep the confidence score as low as possible for cells that do not contain any object. In other words, this reduces the importance of empty cells because background cells are not very informative.

3. **Classification loss:** the last term (2.9) evaluates the quality of the classification estimate, the class prediction estimate, for cells that contain an object.

Thus, classification errors matter in the loss only if an object is located in the given grid cell and bounding box coordinate errors matter only for the one with the highest IoU among all IoU of that grid cell.

The parameters chosen by the authors are $\lambda_{\text{coord}} = 5$, $\lambda_{\text{obj}} = 1$, $\lambda_{\text{noobj}} = 0.5$ and $\lambda_{\text{classes}} = 1$. These choices increase the importance of the predictions of the bounding box coordinates and decrease the confidence for boxes that do not contain objects.

Inference

The inference step refers to the step where a prediction is made using a trained model. Hence, once training is done, the image to which we want to apply object detection, the test image, can be passed through the network. Only one network evaluation is required to detect objects on the test image. Classes, confidence scores and bounding box coordinates can be extracted from the obtained outputs.

[26]Here, the square root value is computed instead of the value. As large bounding boxes have a greater height and width than small ones, squaring these values increases the importance of large boxes and reduces the importance of small ones. Therefore, by computing the square root value instead, the impact of large bounding boxes upon the small ones is reduced. Moreover, using the square root makes the importance of small deviations, between the prediction and the ground truth, less important in large bounding boxes than in small ones.

As explained earlier, at this step, there may be too many bounding boxes for each object. This happens for large objects or for objects near the bounds of multiple cells because each cell is responsible to detect at most one object and multiple cells could detect this same object. So, it is necessary to keep only those that fit perfectly objects. To do so, one can apply non-maxima suppression whose process was described previously, see 2.4.2.

Now there is only a single bounding box per object. Each bounding box is then output with the class of the object it bounds.

Advantages of YOLO

The main advantage of YOLO is its speed. Indeed, the image is given only once to the network for it to apply detection, which implies real-time detection. Therefore, YOLO can be applied to videos in which it is able to detect objects in real time. Moreover, in comparison to other real-time techniques, YOLO is much more accurate. [46]

As explained previously, YOLO is also able to reason about global content of the image when predicting. Indeed, this network sees the whole image during training and encodes contextual information about classes and about their appearance. [37]

Finally, YOLO learns general representations of objects during training. Indeed, if it has been trained on natural images and then tested on artworks, it performs much better than other object detection methods. Thus, YOLO is less likely to make mistakes when switching to new fields or completely unexpected inputs of other datasets. This means that YOLO is generalisable to other domains. [46]

Limitations of YOLO (first version)

YOLO has the advantage of being fast but it also has several drawbacks, such as the difficulty to deal with small objects or objects grouped and close together. Indeed, YOLO assumes at most one object per cell, so if there are several small objects in the same cell, some detections will be missed. Thus, YOLO is not the best object detector for small objects close to each others, such as a group of birds.

We have seen that YOLO is generalisable to other domains. However, it has difficulties in generalising when confronted with objects in other ratios or configurations. Indeed, it is trained to detect on a particular dataset and if objects in other ratios or configurations are provided to the model, it will have difficulties recognising them.

The biggest limitation of YOLO is its difficulty in correctly locating objects, especially small ones and its lack of precision. However, it predicts fewer false positive on the background than other object detection techniques.

Version history

At this point, YOLOv1, the first version of YOLO, introduced by Joseph Redmon et al. in 2016 has been presented. They made their first improvement of YOLO the same year. This second version of YOLO [47] was introduced as “YOLO9000: Better, Faster, Stronger” where 9000 represents the number of categories of objects that YOLOv2 was able to detect. This version has several improvements compared to the first one, such as the improvement of the performance in localising boxes. Following points present some changes of this version.

- A new concept has been introduced: *anchor boxes*. These are predefined bounding boxes of fixed height and width that capture the ideal position of objects to detect based on the provided training data. They aim to help localising bounding boxes. For this purpose, the IoU between the predicted bounding box and the predefined anchor box is computed. The obtained value is used as a threshold. Indeed, if the probability of the detected object is higher than this threshold, then a prediction is made; if it is not, then no prediction is made. These anchor boxes are not randomly initialised here. Since they aim to represent the training data, k-means clustering[27] is applied to these data. Generating boxes using k-means allows the model to start with a better representation and facilitates learning the task. Doing this helped increase the accuracy of this version compared to the first one.

- One limitation of YOLOv1 was its difficulty to adapt to new ratios and configurations. To solve this, *multi-scale training* has been applied in this new version. This consists in randomly resizing the training dataset during the whole training of the model.
- YOLOv2 is also more robust because it has been trained on the merging of two image datasets, one containing the object classes and their respective bounding boxes and the other one containing only the object classes (and thus not containing the corresponding bounding boxes).[28] This means it has been trained on datasets respectively used for an object detection task and for a classification task. The authors have called this way to merge labels of several datasets a *WordTree*. So, this structure is used to combine datasets and then, to jointly train the model on classification and detection.
- About the architecture, another pre-trained model constitutes the backbone[29] of the model. It is called *Darknet-19* and is composed of 19 convolutional layers and of 5 maximum pooling layers. Its architecture is described in [47]. Once this network has been pre-trained, the last convolutional layer is replaced by three 3×3 convolutional layers followed by a final 1×1 convolutional layer to perform feature detection.

In conclusion, these modifications make YOLOv2 faster and more accurate for detection than the first version.

In 2018, Joseph Redmon et al. have improved their algorithm once again and have published the article [48] presenting YOLOv3.

- This point investigates the YOLOv3 architecture. The first layer just rescales the input image for it to have dimensions accepted by the next layers, the convolutional layers. The backbone of this algorithm is *Darknet-53*, an improvement over Darknet-19. It is composed of 53 convolutional layers and of some skip connections, and

[27]The *k-means clustering* algorithm is a method of partitioning data into k groups, called *clusters*. These clusters are constructed in a way that minimises a given distance measure between the points of each cluster. Here, the distance measure used is intersection over union. What is different in our case is that we do not use the (x,y) coordinates of points. In fact, we are working with bounding boxes whose height and width are known. We therefore use k-means on the (w,h) information. In our dataset, we know the height and width of the bounding boxes and, since an anchor box is a candidate to predict these boxes, the more similar the shape (height, width) of the anchor box is to the real box, the better.

[28]Image datasets with object classes and associated bounding boxes are not particularly large compared to those containing only the classes of objects present in the image. Indeed, labelling an image with classes and associated bounding boxes is much more time consuming than just labelling an image with a class. This implies that the datasets available for a detection task are fewer and smaller. However, in general, the more data that are available, the more accurate the trained model will be. This is why the authors combined two datasets: the model can then be trained on a much larger dataset for classification and already have acquired some information that it will then complete in order to perform a detection task.

[29]The *backbone* refers to the feature extractor part of the model.

has been pre-trained on ImageNet. It is publicly available online, see [45] and its architecture is described in [48].

The layers after the backbone (here, Darknet-53), whose goal is to perform detection, have a new structure in this version. Indeed, there are 3 detection branches which aim to detect at different scales and perform feature detection. In other words, YOLOv3 predicts boxes at 3 different scales, a prediction is made at each scale by one of the three branches. These scales are obtained by downsampling the dimension of the input image. The first branch must detect large objects, the second one must detect medium ones and the last one is for small objects. This implies an improvement over YOLOv2. Indeed, YOLOv3 is more accurate at detecting small objects, which was a limitation of the previous version. However, it does not perform better on medium and large objects. Moreover, YOLOv3 is slower than the second version since it has much more layers. Figure 2.27 below illustrates this architecture.

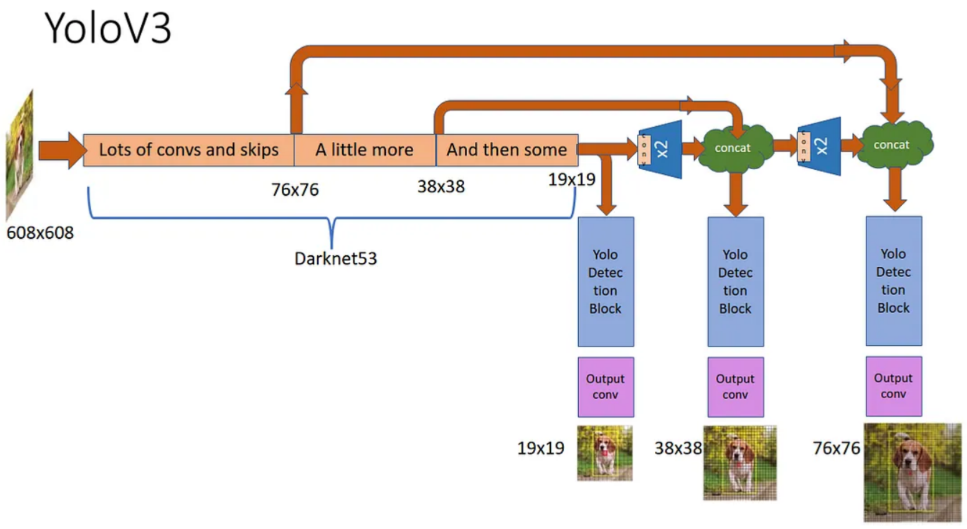


Figure 2.27: Architecture of YOLOv3. © [3]

- The YOLOv3 loss is not exactly the same as before. It is not explicitly given in the article [48], but it is much more complex and it increases the accuracy of the model. For instance, instead of evaluating class predictions through squared error loss, the cross-entropy loss is used. The objectness score, measuring the overlap between bounding box predictions and ground truth boxes, is also improved. In the case where the IoU between these boxes is less than a given threshold, the object prediction is ignored and is considered to be a false positive. Now, in the case where a ground truth object has no bounding box prior, no loss is derived for the predictions of classes and of bounding box coordinates. Only the loss related to objectness is derived, which is done through a cross-entropy loss for each bounding box.

To conclude, this version is more accurate than the previous one but it is slower.

Unlike the three previous versions, the fourth version of YOLO was not written by Joseph Redmon et al. but by Alexey Bochkovski et al. who introduced new improvements, see their article [8] published in 2020.

YOLOv4 has the same backbone as YOLOv3, i.e. Darknet-53. The difference is that some methods have been added during the detector training to increase the detector performance. These methods are classified into two groups called "Bag of Freebies" and "Bag of Specials". The following two points describe them without going into detail.

- **Bag of Freebies:** this kind of methods allows to improve accuracy without increasing the cost of inference.[30] In this case, either the training strategy or the training cost (the cost needed to train the model) is changed. For instance, data augmentation is a bag of freebies method commonly used and here, *mosaic data augmentation*, a new data augmentation method, has been applied. The idea is that 4 training images have been mixed instead of a single image. Another bag of freebies' modification is the change of the bounding box loss. The *Complete Intersection over Union* loss has been used and allows for a more accurate comparison of the height and width of the boxes. We will not describe them but other methods from this family have also been used in YOLOv4.
- **Bag of Specials:** this kind of methods allows to improve significantly the accuracy of object detection but they also imply a small increase in the inference cost. For instance, *Cross-stage partial connections* (CSP) is a skip connection that improves the learning ability of the backbone network, i.e. Darknet-53. As for the previous family of techniques, we will not describe all of them but there are other methods from the bag of specials family that have also been used in YOLOv4.

These changes allow the fourth version of YOLO to be faster and more accurate than the third one. Thus, among all versions of YOLO, this version, YOLOv4, seems to be the best choice to perform object detection.

2.4.3 Region-based CNNs

In this section, another family of object detectors, called region-based convolutional neural networks, is studied. This section is based on what has been seen in the *"Deep learning"* course [37] and on the references [22], [28], [49]. This family of techniques has a strategy different than OverFeat and YOLO. Instead of having a large predefined set of box proposals, region-based CNNs are based on *region proposals* previously extracted from the image. This family includes several architectures such as R-CNN, Fast R-CNN, Faster R-CNN, and Mask R-CNN.

2.4.3.1 R-CNN

R-CNN [23] refers to Region-based CNN. The R-CNN architecture is composed of four distinct parts.

1. **Selective Search:** this algorithm is used to perform a selective search on an input image in order to generate a fixed number of interesting regions, for example 2000. These interesting regions are called *region proposals* or *Regions of Interest (RoI)*. This implies that, unlike in the techniques seen previously, there is not a huge number of regions to analyse and classify. The idea behind this algorithm is that it looks at a given input image through windows of different sizes. Then, for each size, it tries to cluster adjacent pixels on basis of their similarity in texture, colour or intensity.
2. **A pre-trained CNN:** this is the backbone of the architecture. This network is applied independently on each proposed region. First, it must resize each of them to its input dimensions. Then, a forward pass is applied to output features for the region proposals.

[30]Recall that *inference* refers to the step where a prediction is made using a model already trained.

3. **Support vector machine:** this supervised learning method[31] takes as input the features extracted in the previous step and outputs the class prediction. In other words, this method aims to determine whether an object is present in this region proposal.
4. **Linear regression model:** it also takes as input the features extracted in step 2 and outputs the bounding box prediction.

Figure 2.28 illustrates the architecture of this network with two region proposals.

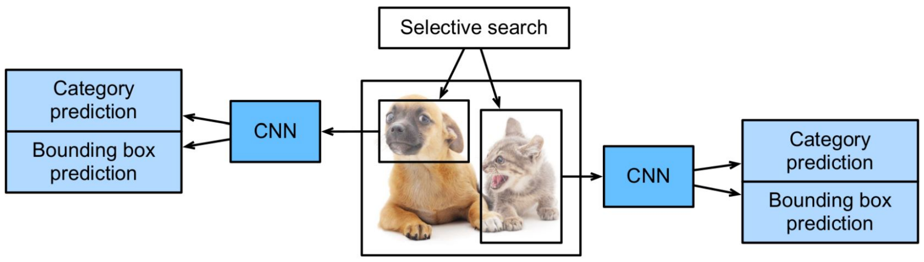


Figure 2.28: Architecture of R-CNN. © [37]

There are some problems with this method. Indeed, it is quite slow to train because if there are 2000 region proposals per image that have to be classified by the pre-trained network, then this implies that 2000 passes through the CNN are needed. In YOLO, only one pass through the network was performed. This implies that it is not a real-time object detector. Another drawback is that the selective search algorithm does not perform any learning because it is a fixed algorithm. This could generate poor region proposals.

2.4.3.2 Fast R-CNN

The goal of Fast R-CNN [22] is to improve one problem of R-CNN, i.e. to perform one single pass in the CNN and not as many as the number of region proposals. Indeed, in R-CNN, it was necessary to independently extract features for each proposed region. For this purpose, the entire image, instead of each proposed region, is given as input to the pre-trained CNN for it to extract features. This implies that the CNN is used only once per image.

After feeding the input image to the CNN, a convolutional feature map is obtained as output. This output feature map can be much smaller than the input image, depending on which CNN is used. Meanwhile, selective search was applied to the input image and has provided proposal regions. Parts of the map corresponding to these region proposals are then fed into the *RoI pooling layer*. As the regions fed in input are of different size, the RoI pooling layer aims to produce feature vectors of fixed size. By the introduction of this layer, another improvement is made compared to R-CNN. The next layer is fed by the RoI feature vector and then splits into two branches, one to predict the class of the region proposal and the other one for the corresponding bounding box prediction. The architecture of this network is illustrated in Figure 2.29.

As said in its name, Fast R-CNN is faster than R-CNN and it also has a better accuracy. However, Fast R-CNN is still linked to the quality of the region proposals from the selective search algorithm. Thus, its performance can be strongly affected by the choice of regions.

[31]See Section 2.1.1.

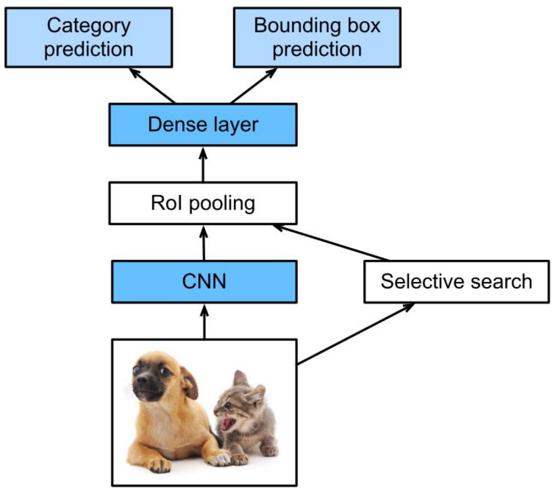


Figure 2.29: Architecture of Fast R-CNN. © [37]

2.4.3.3 Faster R-CNN

Principle

As explained above, R-CNN and Fast R-CNN are linked to the quality of the region proposals because they use the selective search algorithm, which is a fixed algorithm. This drawback is solved in this new version, called Faster R-CNN [49]. Indeed, the *region proposal network*, the algorithm used by Faster R-CNN instead of selective search, learns the region proposals. In fact, this region proposal network aims to reduce the number of proposed regions and to guarantee a more precise object detection.

The architecture is quite similar to that of Fast R-CNN, as can be seen in Figure 2.30. Indeed, the CNN is fed with the input image and outputs a convolutional feature map. Afterwards, an independent network, the *region proposal network*,[32] is applied on the feature map to determine region proposals and is used instead of selective search. Then, the RoI pooling layer reshapes the predicted region proposals and, in each proposed region, allows to perform object classification and to predict the corresponding bounding box, in a similar way to what is done in the two methods seen above.

This new version of R-CNN is much faster than the previous ones so it can be used to perform object detection in real-time.

Remark 2.4.2. In general, YOLO is fast for inference but not the most accurate for object detection, whereas the family of R-CNN methods is slower but often more accurate.

Network design

Initially, Faster R-CNN was presented by the authors with the pre-trained VGG-16 network [54] as the CNN backbone. This network, introduced in 2014, is made of 16 layers associated with weights, i.e. 16 convolutional and fully connected layers, and of 5 max-pooling layers. Figure 2.31 illustrates the architecture of VGG-16.

The authors also tested ResNet-101 instead of VGG-16 as the pre-trained backbone of their model. ResNet-101, introduced in 2015 [29], is made of 5 main convolution blocks with a total of 100 convolutional layers, two pooling layers and one fully connected layer.

[32]In this thesis, we do not go into the detail of the region proposal network architecture.

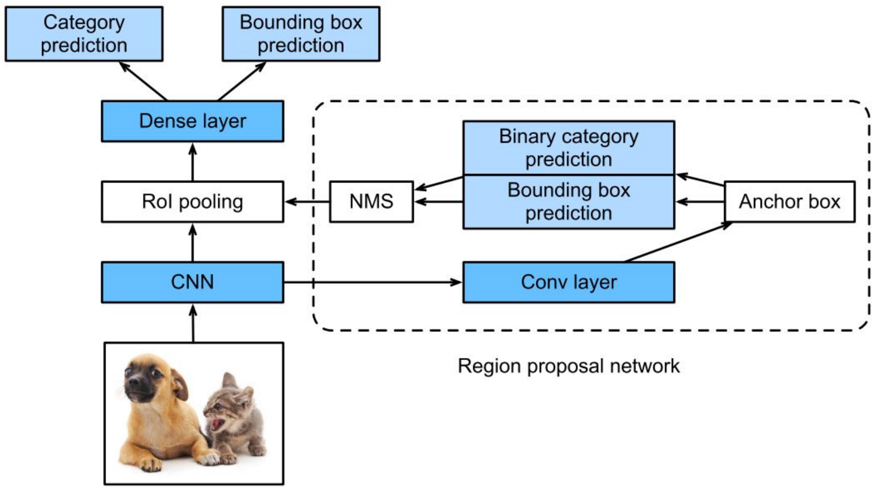


Figure 2.30: Architecture of Faster R-CNN. © [37]

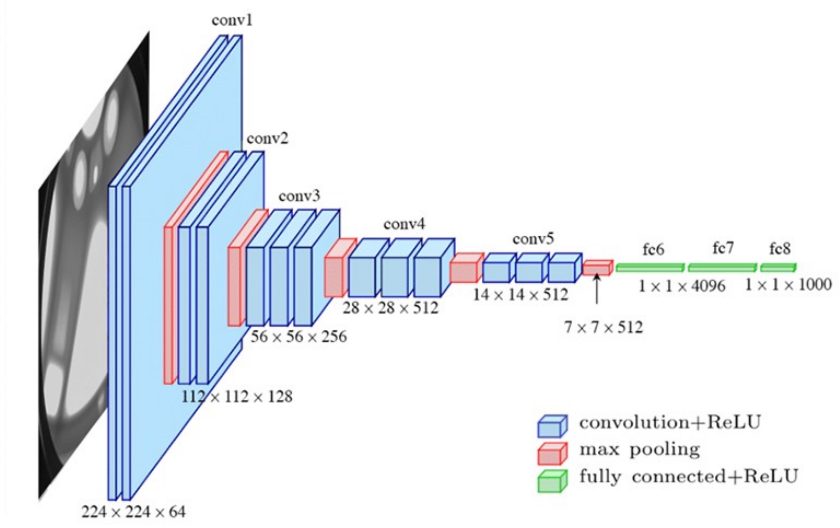


Figure 2.31: Architecture of VGG-16, where "conv" indicates a convolutional layer and "fc" stands for a fully connected layer. © [17]

Using this network as a backbone turned out to perform better than VGG-16. This explains why when Faster R-CNN is used, it is rather employed with ResNet-101 as backbone. Figure 2.32 illustrates the architecture of ResNet-101.

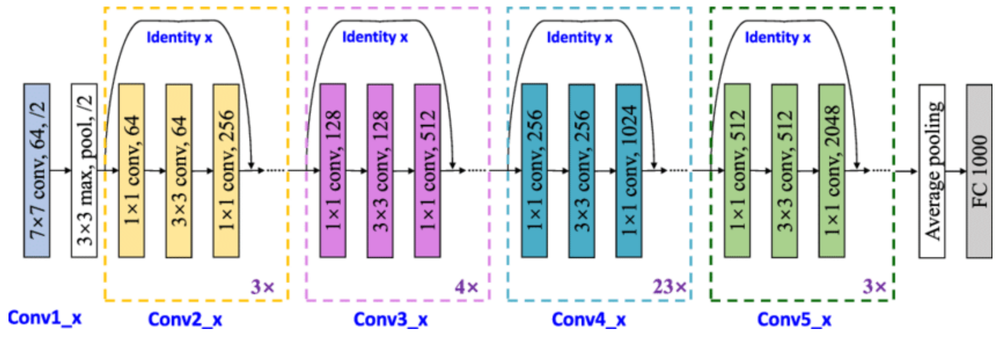


Figure 2.32: Architecture of ResNet-101, where "conv" indicates a convolutional layer and "fc" stands for a fully connected layer. © [14]

Training

As Faster R-CNN is made of two networks, the Region Proposal network (RPN) and the Fast R-CNN part, the loss is a combination of the losses for these tasks. The loss function[33] is given by

$$L = \lambda_1 \, L_{RPN} + \lambda_2 \, L_{FastR-CNN},$$

where λ_1 and λ_2 allow to weigh the two losses and are usually set to 1.

Anchor boxes are used in the Region Proposal network and will influence the corresponding loss. For this purpose, a binary class label (positive or negative), for being an object or not, is associated to each anchor and is defined as follows:

$$\begin{cases} \text{positive} & \text{if the anchor has the highest IoU with a ground-truth box} \\ & \quad \textit{or} \text{ if it has an IoU overlap higher than 0.7 with any ground-truth box} \\ \text{negative} & \text{if the IoU overlap between the anchor and each ground truth box is lower} \\ & \quad \text{than 0.3} \\ \backslash & \text{if none of the two conditions is respected, the anchor does not contribute to} \\ & \quad \text{the training loss function and no label is assigned to it.} \end{cases}$$

For instance, the anchor is said to be positive if the first condition above is verified and is said to be negative if the second condition is verified.

The Region Proposal network loss L_{RPN} is a multi-task loss given by

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

where the following notations are used:

- i: the index of an anchor box in a mini-batch,
- p_i: the predicted probability of anchor i of being positive, with $\{p_i\}$ the outputs of the classification part,
- p_i^*: the binary ground-truth class label, which is equal to 1 if anchor i is positive and 0 if negative,
- $\boldsymbol{t_i}$: a vector representing the 4 coordinates of the predicted bounding box, with $\{\boldsymbol{t_i}\}$ the outputs of the regression part,
- $\boldsymbol{t_i^*}$: a vector representing the 4 coordinates of the ground-truth box for a positive anchor,
- N_{cls}: a normalisation term equal to the mini-batch size,
- N_{reg}: a normalisation term equal to the number of anchor locations,
- λ: a parameter to balance the weights of the classification loss L_{cls} and the regression loss L_{reg}, which is taken equal to 10 by the authors.

The classification loss L_{cls} is the log loss over 2 classes, corresponding to the object and background classes, i.e.

$$L_{cls}(p_i, p_i^*) = -p_i^* \log p_i + (1 - p_i^*) \log(1 - p_i).$$

[33]This part about the loss function is based on the references [49] and [51].

The regression loss L_{reg} is taken as the smooth L_1 loss, which is quite robust to outliers, i.e.

$$L_{reg}(\boldsymbol{t_i}, \boldsymbol{t_i^*}) = \text{smooth}_{L_1}(\boldsymbol{t_i} - \boldsymbol{t_i^*}),$$

in which

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise.} \end{cases}$$

This regression loss is activated in the RPN loss only for positive anchors, which happens when $p_i^* = 1$.

The loss of the Fast R-CNN part, $L_{FastR-CNN}$, is similar to that of RPN. Indeed, there are only two differences between them.

1. In this part, there are no longer anchor boxes. The boxes used are those proposed by the region proposal network. These proposed bounding boxes are considered positive according to the IoU between them and the ground truth boxes. The IoU values for the box to be positive or negative are the same as before.
2. The classification loss L_{cls} is taken as

$$L_{cls}(p_i, p_i^*) = -log(p_i^c),$$

   where p_i^c is the predicted probability that the proposed region belongs to class c, which is the ground truth class.

Remark 2.4.3. The article [40] provides an improved version of Faster R-CNN for detection of vehicles in aerial images. This is interesting for us since in this master's thesis, the dataset we are working on is composed of satellite images. In the article, the authors point out four major problems with vehicle detection, but only one of them concern us. This is the fact that each vehicle occupies only a small part of the image. Indeed, in a satellite image, a vehicle is seen as a small object due to the scale of the picture. This implies that vehicles have a lower resolution and are blurrier than objects in the foreground. Thus, they lack detail and are therefore difficult to categorise.

Now, we investigate a bit their improvement on this problem. As seen previously, a CNN composes the Faster R-CNN method. This model may reduce the ability of distinguishing features for small objects because of the pooling operation. Indeed, these are used to decrease the number of parameters of the network, which can result in a loss of details of feature maps for small objects. For instance, if a vehicle of size 32×32 pixels is fed to four 2×2 pooling operations, the size of the resulting feature map is 2×2 pixels, which is too small to completely describe all information about the vehicle. This implies difficulties in classifying vehicles.

The authors' idea to counter this problem is to perform feature amplification in the last feature map by using bilinear interpolation.[34] [35] Doing this enlarges the feature map, which is useful to restore the detailed information and to improve the ability of feature maps to represent small objects, in this case vehicles. The authors claim that this improves the detection.

[34] We do not go into details of this interpolation, but all details can be found in the original article [40].

[35] Transposed convolution could have been used for this purpose but the authors preferred using interpolation.

2.4.3.4 Mask R-CNN

Principle

Mask R-CNN, introduced in 2018 by He et al. [28], extends the Faster R-CNN model for semantic segmentation, whose principle has been explained in Section 2.3. In a few words, recall that the goal is to classify each pixel of the image in a category and to provide the contours of the object, which is more precise than providing a box around the object, see Figure 2.13. The architecture of Mask R-CNN is shown in Figure 2.33.

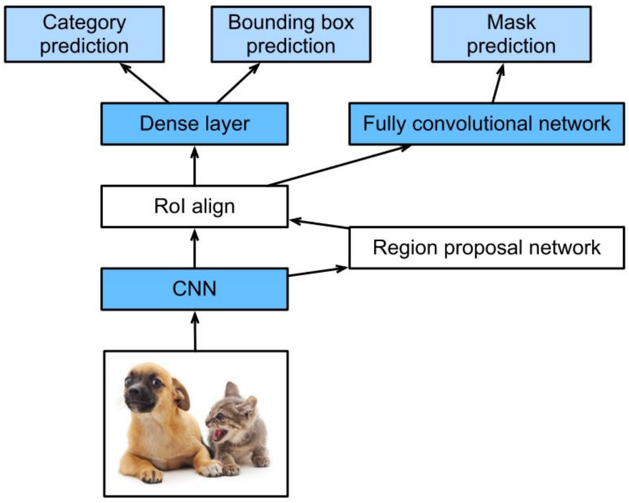


Figure 2.33: Architecture of Mask R-CNN. © [37]

It is very similar to the architecture of Faster R-CNN. The only changes are that the RoI pooling layer has been replaced with the *RoI alignment layer* and that a new branch has been added to predict a segmentation mask using a *fully convolutional network* (FCN). Then, combining object detection (the two first branches) and mask prediction (the third branch) allows to perform instance segmentation. These changes are useful for the following reasons. First, the RoI alignment layer preserves spatial locations, i.e. it properly aligns the extracted features to the input. It also reduces the information loss for this step compared to the RoI pooling layer. Thus, this improves the accuracy of the mask prediction. Second, in parallel to class and bounding box prediction, the new branch for mask prediction predicts K binary masks for each RoI, where K is the number of classes. This binary mask is composed of 1's at the location of the object and 0's elsewhere. These predictions are done independently for each class, which implies that there is no competition among classes. As said before, for each RoI, the mask branch generates a mask for each of the K classes. However, only one mask is kept, the one which is associated to the class predicted by the classification branch.

Mask prediction is performed by a fully convolutional network. So, its purpose is to make predictions for every pixel all at once. The network is made of several convolutional layers, one downsampling step and one upsampling step. Downsampling allows to reduce the input width and height, while upsampling allows to increase them. If neither downsampling nor upsampling is applied and only convolutional layers are used, then the network is very slow. Indeed, if a convolution is applied directly on the input image, then this process will be computationally expensive. It is therefore preferable to reduce the size of the input image before providing it to the first convolutional layer, which is done in the downsampling step. After all convolutional layers, it is then necessary to apply an upsampling step since the output of the network must be an image of the same size as the input image. Indeed, instance segmentation requires to predict values for each pixel of the input image, which corresponds to the output image.

Downsampling is performed using convolutional and pooling layers. If necessary, these layers can also keep the dimension of the input unchanged. Upsampling is performed by a *transposed convolution layer*.

Convolutional and pooling layers can be used to reduce the input width and height. The new layer, transposed convolution layer, allows the inverse of these layers, i.e. it increases the input width and height. For this purpose, a *transposed convolution* is used. This is a convolution where the implementation of the forward and backward passes are swapped.

Figure 2.34 illustrates a simple fully convolutional network (FCN) architecture.

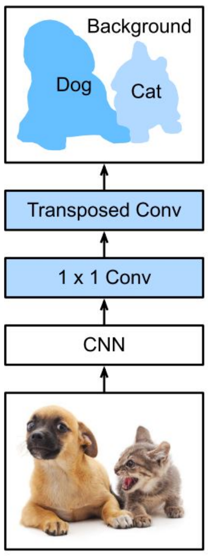


Figure 2.34: Architecture of FCN. © [37]

This architecture is analysed as follows.

- An image is given in input to a pre-trained CNN, which performs the downsampling step and extract features from the image.
- The 1×1 convolutional layer aims to transform the number of channels into the number of categories. For instance, in Figure 2.34, the number of channels is 3 if the input image is represented using the following three main colours: red, green and blue. This number must then be transformed into the number of classes since each class will be represented by a different colour on the output image, i.e. the mask. Here, there are two classes, cats and dogs.
- One (or several) transposed convolution layer(s) can then be applied to the feature map for it to be upsampled to the size of the input image.

Recall that this architecture will be applied to each RoI to predict a segmentation mask and that the dimensions of its output are not fixed but depend directly on those of the input, which can be an image of arbitrary size.

Network design

To illustrate the generality of their approach, the authors who introduced Mask R-CNN have tested it with different backbones:

- ResNet-50-C4: ResNet-50 is a pre-trained CNN of 50 layers, "C4" indicates that the extracted features are those from the final convolutional layer of the 4th block of this model, see Figure 2.35;

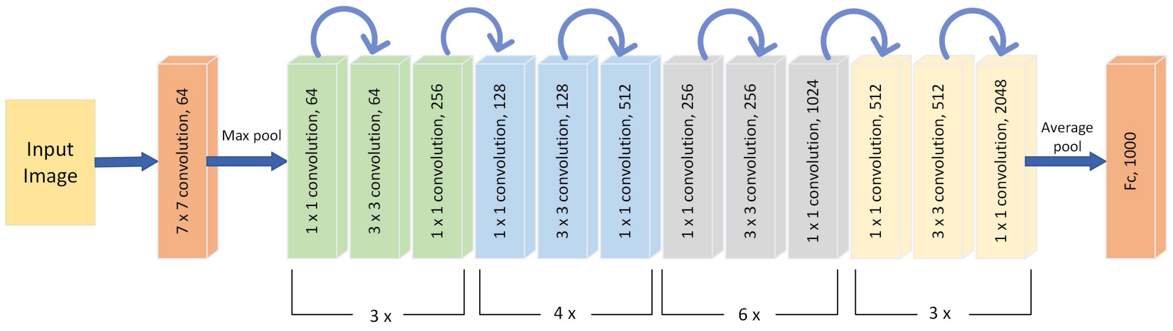


Figure 2.35: Architecture of ResNet-50, where "pool" stands for pooling and "fc" stands for fully connected. © [5]

- ResNet-101-C4: the only difference with ResNet-50-C4 is that the network ResNet-101 is deeper since it has 101 layers instead of 50, as shown previously in Figure 2.32;
- ResNet-50-FPN: Feature Pyramid Network (FPN) is a network that extracts features at multiple scales of an image and that can be used with ResNet-50 as backbone;
- ResNet-101-FPN: ResNet-101 is used as the backbone feature extractor of FPN;
- ResNeXt-101-FPN: exactly the same as above, but using the ResNeXt [59] network instead, which is an improved version of ResNet that reduces the number of hyper-parameters in comparison to ResNet.

All these backbones perform well but the deeper ones, i.e. those with the most layers, seem to be better. Moreover, the use of FPNs offers a gain in speed.

Training

The loss used in Mask R-CNN is a multi-task loss based on the classification loss L_{cls}, the bounding box loss L_{box} and the mask loss L_{mask}. It is defined on each RoI during training as follows

$$L = L_{cls} + L_{box} + L_{mask}.$$

The multi-task loss[36] allows to jointly train for classification, bounding box regression and mask prediction.

In the following, we will use the notation u for the ground-truth class associated to an RoI and v for the ground-truth bounding box regression target associated to an RoI. Each RoI is labelled with this information.

- The classification loss is defined as $L_{cls}(p, u) = -\log(p(u))$, which is the log loss for the true class u.

[36]The classification and bounding box losses are the same as in Fast R-CNN and are explained above, see 2.29.

- The bounding box loss is defined as $L_{box}(\boldsymbol{t^u}, \boldsymbol{v}) = \lambda\, \mathbb{I}_{[u \geq 1]} L_{loc}(\boldsymbol{t^u}, \boldsymbol{v})$, where λ is taken equal to 1 and where

$$L_{loc}(\boldsymbol{t^u}, \boldsymbol{v}) = \sum_{i \in \{x,y,w,h\}} \text{smooth}_{L_1}(t_i^u - v_i).$$

  Above, $\boldsymbol{t^u} = (t_x^u, t_y^u, t_w^u, t_h^u)$ and $\boldsymbol{v} = (v_x, v_y, v_w, v_h)$ where t_i^u for $i \in \{x, y, w, h\}$ represents the values describing the predicted bounding box for class u, while v_i for $i \in \{x, y, w, h\}$ indicates those describing the real bounding box. In the expression above, the smooth L_1 loss is used since this loss is quite robust to outliers, i.e. predictions that are too different from the ground truth. Note that, in the definition of L_{box}, the case where $u = 0$ is not considered since the background classes do not have ground truth bounding boxes, hence the indicator function $\mathbb{I}_{[u \geq 1]}$.

- The mask loss is defined as the average binary cross-entropy loss:

$$L_{mask} = -\frac{1}{m^2} \sum_{1 \leq i,j \leq m} [y_{i,j} \log y_{i,j}^u - (1 - y_{i,j}) \log(1 - y_{i,j}^u)],$$

  where
  - $m \times m$ corresponds to the size of the region of interest,[37]
  - $y_{i,j}$ provides the class of cell (i, j) in the true mask,
  - $y_{i,j}^u$ represents the prediction done for the cell (i, j) in the mask learned for the ground-truth class u.

At this point, we have seen how Mask R-CNN works and that it is designed for instance segmentation. However, it can be used to perform only object detection, which is precisely what interests us in this master's thesis. Indeed, the full model must be trained, but only the classification and bounding box predictions will be used. The third output predicting the mask is disregarded. The authors who introduced Mask R-CNN have tested this and it outperforms other object detection methods.

However, the dataset we are working with does not contain the masks associated to the objects in images. It is therefore not possible to train the full model, but the authors have also investigated this situation. They trained a model without the third branch, i.e. the one corresponding to mask prediction. According to them, even if this outperforms other object detector thanks to the use of the RoI alignment layer, this performs less effectively than when masks were available. Thus, they claim that multi-task learning seems to be beneficial for box detection. So, in our case, using Mask R-CNN without the mask branch is equivalent to Faster R-CNN where the RoI pooling layer has been replaced by the RoI alignment layer.

To conclude about Mask R-CNN, it is a method for instance segmentation which can also perform a real-time object detection. Indeed, it is fast for training and for inference. This method has other advantages since Mask R-CNN has a great accuracy, it can also be easily generalisable to other tasks and it is quite intuitive. Indeed, this unified model performs simultaneously mask prediction, object classification and bounding box prediction since all three tasks have been trained jointly.

[37]In Mask R-CNN, the region of interest is square.

Part II

Practical results

Chapter 3

Methodology

This chapter first presents LS-SSDD-v1.0, i.e. the dataset selected by the first student who worked on this topic of ship detection using deep learning. The different subsets created from this dataset are then presented, along with their purpose in this practical part. The last section describes the structure of Part II about "Practical results".

3.1 LS-SSDD-v1.0 dataset

Large-Scale SAR Ship Detection Dataset-v1.0 or LS-SSDD-v1.0 has been presented in the article [61] in 2020. This dataset has been designed for small ship detection on large-scale backgrounds. It includes 15 large-scale SAR images collected from the Sentinel-1 satellite in the interferometric wide swath mode using VV polarisation. These images correspond to ground range multi-look detected (GRD) data. They have been uniformly resized to a format of $24,000 \times 16,000$ pixels. As these images are particularly large, they have been cut into 9,000 sub-images of 800×800 pixels. This results in a larger dataset and this facilitates the training of the deep learning algorithm, partly because it reduces the limitations of the GPU memory. The rest of this section provides more information about these images, the cutting into sub-images and the advantages of this dataset.

The large-scale images correspond to different geographical areas of the Earth, see Figure 3.1, that have been selected because they include busy ports, straits, river areas, etc. Each large-scale image of $24,000 \times 16,000$ pixels was cut into 20 rows and 30 columns, leading to 600 sub-images of 800×800 pixels. This resulted in a total of 9,000 sub-images. These were named in the format $N\_R\_C.jpg$ where N corresponds to the number of the large-scale image, R corresponds to the row of the sub-image and C corresponds to its column. Figure 3.2 illustrates this cutting for large-scale image number 11. The sub-images are indicated and numbered in yellow and range from image 1_1.*jpg* to 1_30.*jpg*, from 2_1.*jpg* to 2_30.*jpg*, ..., and from 20_1.*jpg* to 20_30.*jpg*.

To obtain the ground truth bounding boxes of the ships appearing in the images, the 9000 sub-images were annotated by SAR experts based on AIS messages and using Google Earth. The latter program was used to make corrections to the information provided by the AIS messages. For example, it has been used to identify ships of less than 30 tons, which generally do not have AIS, or to make corrections when islands or reefs were mistakenly labelled as ships. The annotations were made using the LabelImg[1] program. The bounding

[1]This graphical tool for image annotation can be downloaded from the website [34].

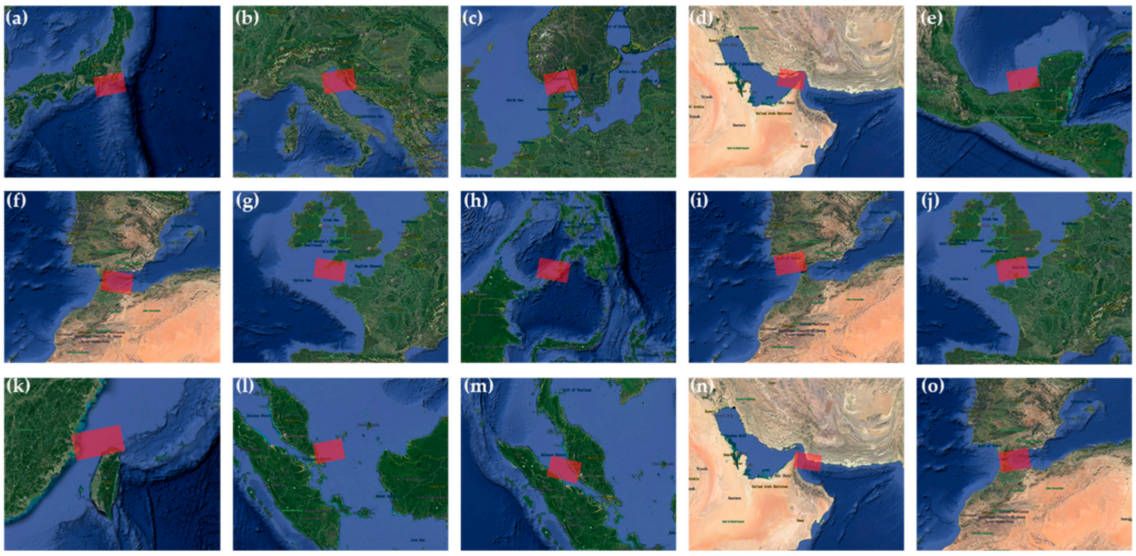


Figure 3.1: Coverage areas of the 15 large-scale images. (a) Tokyo Port; (b) Adriatic Sea; (c) Skagerrak; (d) Qushm Islands; (e) Campeche; (f) Alboran Sea; (g) Plymouth; (h) Basilan Islands; (i) Gulf of Cadiz; (j) English Channel; (k) Taiwan Strait; (l) Singapore Strait; (m) Malacca Strait; (n) Gulf of Oman; (o) Gibraltarian Strait. © [61]

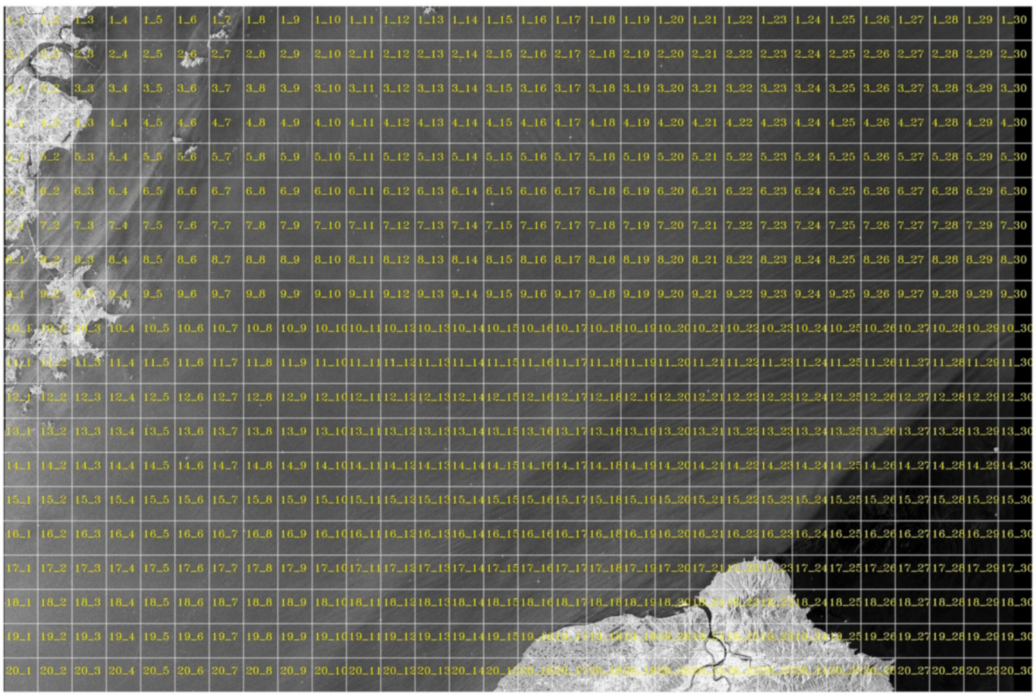
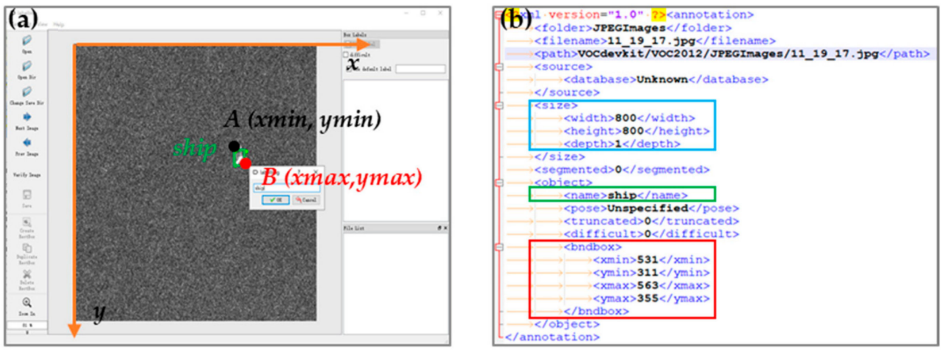


Figure 3.2: Illustration of the cutting in sub-images of the large-scale image number 11. © [61]

boxes are annotated using the PASCAL VOC format, i.e. using the coordinates of the top left vertex $(xmin, ymin)$ and those of the bottom right vertex $(xmax, ymax)$. The position of a bounding box is then noted as $(xmin, ymin, xmax, ymax)$. Figure 3.3 (a) illustrates the annotation of a box, with the x and y axes shown in orange. As illustrated in Figure 3.3 (b), the program then creates an *xml* file containing the characteristics of the image, displayed in the blue rectangle, i.e. its size of 800 x 800 pixels; the category of the object detected in the green rectangle, i.e. a ship, and the position of the bounding box in the red rectangle.



```xml
<?xml version="1.0" ?><annotation>
    <folder>JPEGImages</folder>
    <filename>11_19_17.jpg</filename>
    <path>VOCdevkit/VOC2012/JPEGImages/11_19_17.jpg</path>
    <source>
        <database>Unknown</database>
    </source>
    <size>
        <width>800</width>
        <height>800</height>
        <depth>1</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>ship</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>531</xmin>
            <ymin>311</ymin>
            <xmax>563</xmax>
            <ymax>355</ymax>
        </bndbox>
    </object>
</annotation>
```

Figure 3.3: (a) Example of the annotation process using LabelImg. (b) Example of a *xml* file that contains the annotation of a ground truth bounding box in Pascal VOC format. © [61]

The authors who published this dataset claim that it has several advantages over the other datasets available before its publication.

- **Large-scale background:** Each large-scale SAR image covers a swath width of 250km. The advantage of using such images is that they cover large areas of the Earth, providing various detection backgrounds. Such features are particularly interesting when monitoring ships.

- **Small ship detection:** In the context of deep learning, the term "small ship" does not refer to ships that are physically small in terms of meters, but rather to ships that occupy a minor number of pixels in the image. In other words, small ships are ships occupying a small proportion of the image. In their article presenting the LS-SSDD-v1.0 dataset, the authors illustrate this difference between small and large ships using Figure 3.4. Ships in the images on the left occupy a small proportion of the image, unlike ships in the images on the right.

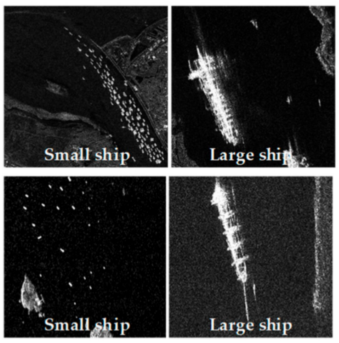


Figure 3.4: Small ships and large ships in SAR images. © [61]

Datasets already exist containing images of ships of different scales, but not containing only small ships. LS-SSDD-v1.0 is therefore a dataset dedicated to them. In fact, according to the article, of the 6015 ships identified in their dataset, 6003 are considered to be small ships, 12 medium ships and 0 large ships.

- **Abundant pure backgrounds:** In the datasets that already exist, images all include at least one ship. They are therefore all positive images and there are no negative images, i.e. images that do not include any ships, because these are artificially abandoned. However, such practice is controversial in application. Real SAR images contain few ships and are often pure background. Hence, human intervention in the removal of negative images therefore does not reflect the usual properties of these images. In addition, a deep learning algorithm could be less effective at learning pure background characteristics, thereby increasing the number of false alarms, i.e. the number of false detections identifying bright dots as ships when they correspond to urban areas, agricultural regions, mountain areas, etc. The creators of LS-SSDD-v1.0 therefore decided to keep all the sub-images without artificially filtering the negative ones. In addition, although using so many pure background samples is time-consuming,[2] they consider that it is still worthwhile in terms of performance measures. In fact, LS-SSDD-v1.0 contains 8 types of pure background that do not include ships: pure ocean surface, farmlands, urban areas, Gobi Desert, remote rivers, villages, volcanos and forests. Figure 3.5 illustrates it.

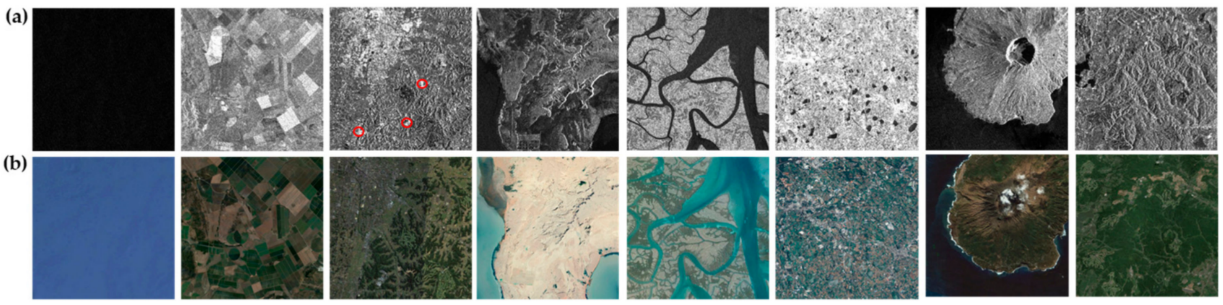


Figure 3.5: Abundant pure backgrounds of SAR images in LS-SSDD-v1.0. (a) SAR images; (b) optical images. These images correspond respectively to: sea surface, farmlands, urban areas, Gobi, remote rivers, villages, volcanos, forests. © [61]

- **Numerous and standardized research baselines:** They compared many different deep learning methods on their dataset under the same training conditions by taking almost the same hyperparameter configuration (it is not possible to have the same ones everywhere, but they tried to take them as close as possible). In particular, they studied Faster R-CNN and YOLOv3. They concluded that Faster R-CNN worked quite well (mAP of 74.80% and F1-score of 0.76) and that YOLOv3 was one of the worst methods (mAP of 24.63% and F1-score of 0.40) as it is not a suitable technique for detecting small ships.

3.2 Subsets

The LS-SSDD-v1.0 dataset described above includes only 1859 images containing at least one ship out of 9000. In general, it is recommended to train an algorithm on a balanced dataset, i.e. a dataset containing as many positive as negative images. [4] [30] In particular, this applies to YOLO. [9] A subset was therefore created by selecting the 1859 positive images and randomly selecting 1859 negative images. This set was called PN. In order to compare the performance measures of the methods studied according to the dataset on which it was trained, two other datasets were created: dataset P, which is composed exclusively of the 1859 positive images, and dataset P2N, which contains twice as many negative images as positive images. The table below summarises the information about these 4 datasets, specifying the total number of images and the number of positive and negative images.

[2]Using more samples slows down the training phase of the algorithm but the testing phase is not affected.

	Total	**Positive**	**Negative**
Complete	9000	1859	7141
P	1859	1859	0
PN	3718	1859	1859
P2N	5577	1859	3718

These datasets have different purposes:

- P dataset: it is used for model optimisation. The train - validation - test method was used. Thus, the set was randomly divided into 3 subsets of train, validation and test, respectively of 80%, 10% and 10% of the size of the total set. It is the validation set that is used to tune the model hyperparameters based on several performance measures. This leads to the choice of a final model, i.e. with fixed hyperparameter values.
- PN dataset: its use is the same as that of P dataset described above, but only for YOLOv3. The reason why it is only used for this method will be explained later.
- P-PN-P2N-complete datasets: they are all used for model assessment. Once the model has been optimised on the P dataset, i.e. the best hyperparameters have been obtained, this final model is trained independently on the other datasets (on the training subset of PN, P2N and complete). The same 80/10/10 splitting as described before is used. Then, for each dataset, the performance measures of the model are computed on the test subset. The resulting performances of the different datasets can then be compared. This allows us to determine, for example, whether training exclusively on positive images leads to better performance than training on a dataset with a certain proportion of negative images.
- PN dataset: for each method, once the best model has been determined, this dataset is used to optimise the thresholds, which completes model optimisation. Indeed, confidence and intersection over union (IoU) thresholds can be modified to obtain better performance measures.
- PN dataset: this is used to assess and visually compare the optimised models of YOLOv3, YOLOv4 and Mask R-CNN. The idea of the visual comparison is to display the bounding boxes on the images of the test set in different colours depending on whether they are true positive, false positive or false negative and then compare the detection difficulties of each method.

3.3 Structure of the part "Practical results"

In this master's thesis, the following three methods are studied: YOLO version 3, YOLO version 4 and Mask R-CNN. A first chapter will be devoted to their model optimisation, in which their visual difficulties will also described. The second will consist of a comparison of the optimised models. To this end, a comparison of their performance measures and their difficulties will be presented. In this chapter, each optimised model will also be applied to a new large-scale SAR image to provide inference on external data. The two following chapters will explain the implementation done within the scope of this master's thesis, as well as the technical problems encountered. Next chapter will conclude the results obtained in this practical part. The last chapter will explain the limitations of our study.

Chapter 4

Optimisation of the models

In this chapter,[1] we first present the different methods employed for computing performance measures, see Section 4.1. Then, we explain the model optimisation of the three methods of interest, YOLO versions 3 and 4, and Mask R-CNN, see Sections 4.2, 4.3 and 4.4, respectively. Now, we briefly provide some information about the process of model optimisation.

For each method, model optimisation first involves optimising the hyperparameters. For this purpose, the hyperparameters are separated into two categories (global and local) because some have a more general influence on training and on the resulting performance measures, while others, the local ones, have a less general influence and should be tuned after the values of the global hyperparameters have been set. Thus, the influence of the global hyperparameters is first studied in order to set their values to obtain the best possible performance measures. After that, the local hyperparameters can be studied and set to potentially improve the performances. Some hyperparameters, such as the learning rate, can be considered both global and local.

It is worth noting that, to optimise the three models, the principle of early stopping was applied, i.e. training was stopped before the last iteration to prevent overfitting. To that aim, the model providing the best mAP was selected.

Once the model is optimised, it can be assessed using performance measures and the visual difficulties can be analysed. However, there remain some hyperparameters that can be optimised: the thresholds on IoU and confidence, set by default at 0.5 and 0.3. By modifying them, it is possible to increase the performance measures and to reduce some visual difficulties.

This final step of model optimisation is carried out on PN dataset because it is usually better to generalise from a dataset with as many positive as negative images. In addition, as our study is based on the analysis of visual difficulties, a set without negative images would not sufficiently reflect reality. Conversely, analysing visual difficulties on a set containing many more negatives than positives might not be sufficiently representative of detection difficulties.

We also studied the influence of modifying hyperparameters on training time. However, we obtained few conclusions on this. Therefore, in this chapter, when the influence on training time is not mentioned, this means that no conclusion was reached.

[1]Each SAR image used in this chapter for illustration comes from the LS-SSDD-v1.0 dataset.

4.1 Methods for computing performance measures

For each method, the performance measures used for model optimisation are F1-score, precision, recall and mAP. Our aim is mainly to maximise the F1-score and mAP since they combine precision and recall, which are nevertheless given for information.

To optimise the YOLO models, we use the performance measures provided by Darknet,[2] the framework use to train YOLOv3 and YOLOv4. With Darknet, precision, recall and F1-score are computed from the confidence theshold that maximises the F1-score. In addition, the mAP is calculated by default without interpolation according to the PASCAL VOC 2010-2012 convention, i.e. it is the exact value of the area under the curve that is computed and not an approximation (other conventions approximate this area).

As performance measures provided by Darknet are only available for YOLO, other performance measures were required to optimise Mask R-CNN. In addition, to properly compare YOLOv3, YOLOv4 and Mask R-CNN, the same method to compute the performance measures should be used. To that aim, a new method was implemented, whose calculation of performance measures is slightly different to that of Darknet: instead of computing precision, recall and F1-score from the confidence threshold that maximises the latter, these are calculated using the confidence threshold set for model optimisation. Then, as with Darknet, the mAP is calculated without interpolation.

In the following, we will refer to these performance measures obtained by an independent program as "external performance measures".

4.2 YOLO version 3

This section presents the hyperparameters of YOLOv3, the optimisation of the model and the visual difficulties encountered by the optimised model.

4.2.1 Hyperparameters

This section lists the hyperparameters used in YOLOv3. As they have already been described in detail in Section 2.2.4.1, their purpose is only briefly recalled. In addition, their default value in YOLOv3 are given.

- **Batch size:** this is the size of a mini-batch, i.e. the number of training images used during one iteration. By default, this is set to 64.
- **Subdivisions:** this is the number into which the mini-batch should be divided. This is an important hyperparameter when memory is limited. By default, this is set to 16.
- **Image size:** this is the size to which images must be rescaled once they have been fed to the algorithm. The LS-SSDD-v1.0 images are 800×800 pixels. By default, YOLO resizes them to a size of 416×416.
- **Learning rate:** it controls the step size used during the training process and it is set to 0.001.
- **Burn in:** this is set to 1000. This means that at the 1000^{th} iteration, the mean average precision (mAP) starts to be computed.

[2]Darknet is an open-source framework for training neural networks that is written in C and CUDA, and includes the implementation of several architectures, including YOLO. In addition, it is fast, easy to install, and supports CPU and GPU computation. [45]

- **Momentum:** during training, it gives more importance to latest updates. This is set to 0.9.
- **Decay:** it penalises model complexity and is set to 0.0005.
- **Steps - Scales:** this modifies the value of the learning rate according to the iteration. By default, steps is fixed to 6000-6750 and scales to 0.1-0.1. This means that the value of the learning rate is multiplied by 0.1 at the 6000th iteration and that this new value of the learning rate is itself multiplied by 0.1 at the 6750th iteration.
- **Iterations:** this is the number of iterations.

4.2.2 Hyperparameters optimisation

This section covers hyperparameters optimisation, with the first part focusing on global hyperparameters and the second on local ones.

4.2.2.1 Global hyperparameters study

This section explains how we studied the influence of the global hyperparameters in order to decide how to set them. The global hyperparameters are batch size, subdivisions, image size, learning rate and burn in.

These hyperparameters have been studied on P and PN datasets. As mentioned earlier, it is generally preferable to train on a balanced dataset, hence the use of PN. The P dataset was used to compare whether the training and hyperparameters behaved similarly on another dataset, in this case one containing no negative image.

Many tests with different global hyperparameter values were performed. These tests were done using 10,000 iterations. This enabled to study the behaviour of each hyperparameter and to select the best value based on the performance measures (F1-score, precision, recall and mAP).

Subdivisions

To determine its influence, different values of subdivisions have been tested: 4, 8, 16 (default value), 32 and 64, while fixing the other hyperparameters. Table 4.1 explains which combinations of values were tested according to the image and batch sizes (the other hyperparameters are set to their default values).

Image size / Batch size	16	32	64
416×416	4, 8, 16	8, 16, 32	16, 32
608×608	8, 16	16, 32	32, 64

Table 4.1: The tests in grey were not performed on the PN dataset because of their long training time. Indeed, training these 4 additional tests would have taken around 70 hours. The analysis of the behaviour of subdivisions on PN dataset is therefore based solely on the tests shown in black.[3]

For instance, for an image size of 416×416 and a batch size of 16, subdivision values of 4, 8 and 16 were tested. Values below 4 could not be tested because training uses too much memory, which causes the training to crash. Values greater than the batch size can not be tested either since subdivisions must be taken smaller or equal to the batch size.

[3]At the moment of these practical tests, training was slowed down due to technical problems, see Chapter 7, which explains the long training time. If these tests were performed when the problems were resolved, they would probably have taken around 40 hours.

Similarly, for an image size of 608×608 and a batch size of 32, subdivision values of 16 and 32 were tested. Values below 16 could not be tested due to memory limitations and subdivisions could be taken at most equal to the batch size, i.e. 32.

The table below illustrates the influence of subdivisions when it <u>decreases</u> while the other hyperparameters are fixed. First, the same behaviour is observed on P and PN datasets. On the validation and test sets, the performance measures (F1-score, recall and mAP) increase when the value of subdivisions decreases. On the test set, the precision decreases, whereas, on the validation set, no conclusion is reached for it.

		F1-score	precision	recall	mAP
Validation set	PN dataset	↗	–	↗	↗
	P dataset	↗	–	↗	↗
Test set	PN dataset	↗	↘	↗	↗
	P dataset	↗	↘	↗	↗

In addition, we found that when the value of subdivisions decreases, the training time tends to increase from a few minutes to a few hours. Furthermore, when subdivisions and batch size are set to the same value, the mAP curve is sometimes very unstable. Figure 4.1 illustrates two training graphs, one with an unstable mAP curve and the other with a stable curve. The graph with the best behaviour is the one whose mAP curve oscillates less and stabilises after a sufficient number of iterations.

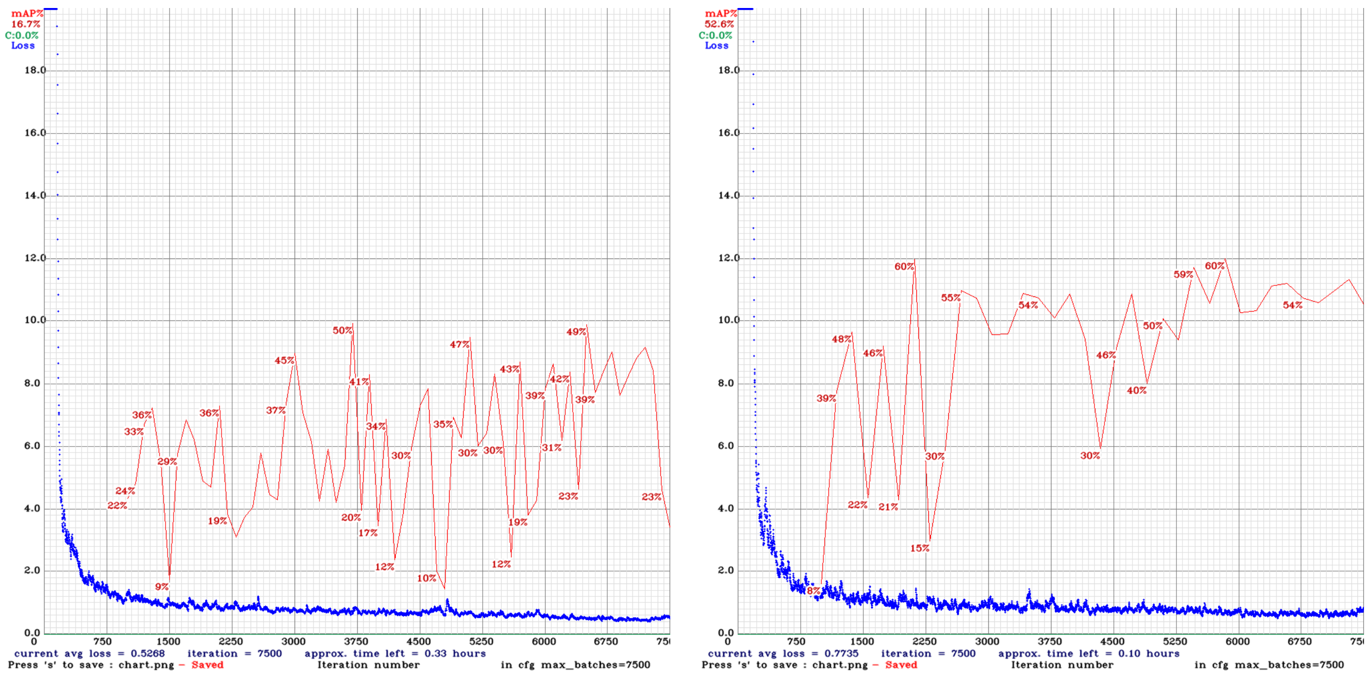


Figure 4.1: On the left: graph of a training with an unstable mAP curve. On the right: graph of a training with a stable mAP curve.

In conclusion, for fixed image and batch sizes, **the smallest possible value of subdivisions should be used**. Thus, for example, for an image size of 416×416 and a batch size of 16, subdivisions should be set to 4. Similarly, for an image size of 608×608 and a batch size of 32, subdivisions should be set to 16.

Among these conclusions, the only one that is consistent with the theory is that small values of subdivisions use more memory than large values. Indeed, values that are too small lead to crashes because of memory issues. The other conclusions are not theoretically expected. A small value should imply faster training, but the opposite has been observed. In practice, performance measures are better when the value of subdivisions is taken as small as possible, but this is not mentioned in theory.

Batch size

To determine its influence, different values of the batch size have been tested: 16, 32, 64 (default value) and 128, while fixing the other hyperparameters. Table 4.2 explains which combinations of values were tested according to the image size and the subdivisions (the other hyperparameters are set to their default values).

Image size / Subdivisions	8	16	32
416×416	16, 32	16, 32, 64	32, 64, 128
608×608	–	16, 32	32, 64

Table 4.2: The 12 tests shown in the table have been performed on the P dataset, but those in grey were not performed on the PN dataset because of their long training time. Indeed, training these 2 additional tests would have taken around 37 hours because of the technical problems, as previously mentioned.

For instance, for an image size of 416×416 and subdivisions set to 16, values equal to 16, 32 and 64 were tested for the batch size. Values over 64 could not be tested because training uses too much memory, which causes the test to crash. Values smaller than the batch size can not be tested either since the batch size must be taken larger or equal to subdivisions. Similarly, for an image size of 608×608 and subdivisions set to 32, the batch sizes of 32 and 64 have been tested.

The table below illustrates the influence of the batch size when it increases while the other hyperparameters are fixed. The $*$ symbol means that this is true except for some tests with a batch size of 128. Almost the same behaviour is observed on P and PN datasets. On the validation and test sets, the performance measures (F1-score, recall and mAP) increase when the batch size increases, except for some tests with a batch size of 128.

		F1-score	precision	recall	mAP
Validation set	PN dataset	$\nearrow$	–	$\nearrow$	$\nearrow$
	P dataset	$\nearrow$	–	$\nearrow$	$\nearrow$ $*$
Test set	PN dataset	$\nearrow$	–	$\nearrow$	$\nearrow$
	P dataset	$\nearrow$	–	$\nearrow$	$\nearrow$ $*$

In addition, it was found that when the batch size increases, the training time increases significantly by several hours. Furthermore, as mentioned previously, when subdivisions and batch size are fixed at the same value, the mAP curve is sometimes very unstable.

In conclusion, for a fixed image size and subdivisions, **the largest possible batch size should be used**. However, since taking on a large batch size implies a much longer training, a compromise between training time and performance measures should be taken.

For example, a good trade-off is to use a batch size at most equal to 64 for an image size of 416×416 and at most equal to 32 for an image size of 608×608, because using a higher batch size requires at least 13 hours of training.

The conclusions obtained here reflect what is expected from theory. Indeed, a larger batch size implies more accurate estimations, but a slower training time.

Image size

To determine its influence, different values of the image size have been tested: 416×416 (default value), 608×608 and 800×800, while fixing the other hyperparameters. It is not possible to test larger image size since images of LS-SSDD-v1.0 dataset have a size of 800×800. Table 4.3 explains which combinations of values were tested according to the batch size and the subdivisions (the other hyperparameters are set to their default values).

Batch size / Subdivisions	8	16	32	64
16	416, 608	416, 608, 800	–	–
32	–	416, 608	416, 608, 800	–
64	–	–	416, 608	608, 800

Table 4.3: The tests displayed in black and gray (except those in blue) were performed on the P dataset. The tests displayed in black and blue (except those in grey) were performed on the PN dataset. Training the 7 additional tests would have taken too much time.

For instance, for a batch size of 16 and subdivisions set to 16, image sizes of 416×416, 608×608 and 800×800 were tested. Similarly, for a batch size of 64 and subdivisions set to 32, image sizes of 416×416 and 608×608 were tested. An image size of 800×800 can not be tested here because the batch size must be equal to subdivisions, otherwise it crashes because it requires too much memory.

The table below illustrates the influence of the image size when it <u>increases</u> while the other hyperparameters are fixed. The behaviours observed on P and PN datasets are very similar. On the validation set of P and PN datasets, when the image size increases, the mAP increases, but the F1-score decreases. This is also observed on the test set of PN dataset. In each case, the mAP increases when the image size increases.

		F1-score	precision	recall	mAP
Validation set	PN dataset	$\searrow$	$\nearrow$	$\searrow$	$\nearrow$
	P dataset	$\searrow$	–	–	$\nearrow$
Test set	PN dataset	$\searrow$	$\nearrow$	$\searrow$	$\nearrow$
	P dataset	–	–	–	$\nearrow$

In addition, it was found that when the image size increases, the training time increases significantly by several hours. Furthermore, as previously mentioned, when subdivisions and batch size are fixed at the same value, the mAP curve computed during training is sometimes very unstable.

In practise, training is longer when the image size is larger, which was expected from theory. Moreover, this should imply a better accuracy, the decrease of the F1-score is then surprising.

In conclusion, for a fixed batch size and subdivisions, **using an image size of 608×608 seems to be a good trade-off**. Indeed, taking 416 × 416 leads to a poorer mAP, while taking 800 × 800 leads to a better mAP but a poorer F1-score. Furthermore, taking an image size of 800 × 800 implies a much longer training, whose mAP curve may be unstable.

Trade-off between batch size and subdivisions

At this point, it has been determined to take an image size of 608×608, the smallest possible subdivision and the largest possible batch size. This leads to different possibilities, which are illustrated in the table below.

Image size	Batch size	Subdivisions
	64	32
608 × 608	32	16
	16	8

Then, we must answer the following question: is it better to take the smallest subdivision value with its associated batch size, i.e. a batch of size 16 with 8 subdivisions, or to take the largest batch size with its associated subdivision value, i.e. a batch of size 64 with 32 subdivisions, or a trade-off between these choices by taking a batch of size 32 with 16 subdivisions? Since the second option takes too much time to train (around 12 hours) and that the third one leads to better performance measures than the first one, the third option seems to be the best one. It takes around 6:45 hours to train. The following table summarises the conclusions on the hyperparameter values that have been reached.

Image size	Batch size	Subdivisions
608 × 608	32	16

Learning rate

To determine its influence, different values of the learning rate have been tested: 0.0001, 0.0005, 0.001 (default value), 0.0025 and 0.005, while fixing the other hyperparameters to their default values. Table 4.4 presents the models on which these values have been tested and then compared.

Batch size	Subdivisions	Image size
16	4	416 × 416
16	8	608 × 608
32	8	416 × 416
32	16	608 × 608
64	16	416 × 416

Table 4.4: The 2 tests displayed in blue and gray were performed on the P dataset. Tests displayed in black and gray were performed on the PN dataset.

Following conclusions have been reached:

- a learning rate of 0.0001 always leads to worse performance measures;
- there is not much difference between learning rate values set to 0.0005, 0.001 and 0.0025, even if this last value seems generally better than the others;

- a learning rate set to 0.005 either leads to the best performance measures, either to unstable mAP and/or loss curves. A training with an unstable mAP curve has already been illustrated in Figure 4.1 and one with an unstable loss curve is illustrated in Figure 4.2.

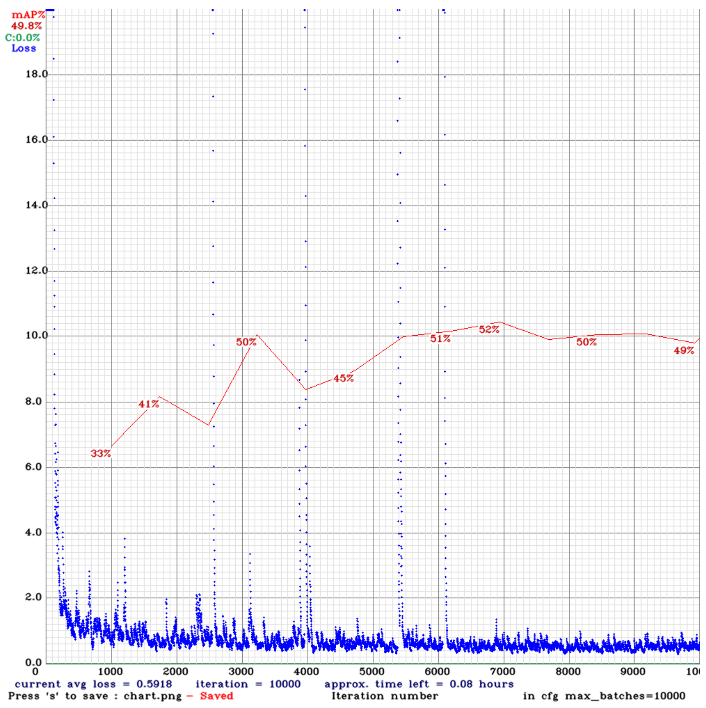


Figure 4.2: Graph of a training with an unstable loss curve.

- there is a small impact of a few minutes on training time, but there is no general tendency.

According to theory, a large learning rate implies faster training, which has not been observed in practise, and if it is too large, the algorithm can diverge and provide a sub-optimal solution. Moreover, a large learning rate is more appropriate when using a large batch size. Thus, the learning rate will be reviewed and set later, when the batch size will be fixed. The study conducted here permits to define a range of values to be tested after setting the batch size. These values are 0.0005, 0.001, 0.0025 and 0.005.

Burn in

To determine its influence, different values: 500, 1000 (default value), 2000 and 3000 have been tested for the burn in while fixing the other hyperparameters to their default values. Table 4.5 presents the models on which these values have been tested and then compared.

Batch size	Subdivisions	Image size
16	4	416×416
32	16	608×608
32	8	416×416
64	16	416×416

Table 4.5: The two first models have been used to compare the burn in values on P dataset and the two following models have been used for PN dataset.

As the results are very similar and no conclusion is reached, this hyperparameter is fixed to its default value, i.e. to 1000.

Summary

The following table summarises which value or range of values should be selected for each global hyperparameter according to the conclusions of our study.

Image size	Batch size	Subdivisions	Learning rate	Burn in
608×608	32	16	0.0005, 0.001, 0.0025 or 0.005	1000

In next sections, image size, batch size, subdivisions and burn in are set to these values.

4.2.2.2 Local hyperparameters study

This section explains how we studied the influence of the local hyperparameters in order to decide how to set them. The local hyperparameters are the learning rate, momentum, decay and steps-scales. The learning rate is a global hyperparameter since it has a large influence on training and on the resulting performance measures, and it was thus studied in previous section. It is also a local hyperparameter because it has less influence once the other hyperparameters have been fixed, in particular the batch size, and is therefore studied in this section.

Each test of this section was done using 7500 iterations. They were all performed on P dataset only and no longer on PN, because the conclusions obtained in the previous section were the same for both sets and because the performance measures were better on P than on PN.

For each local hyperparameter, several values were tested, four tests for each of them were performed and the resulting performance measures (F1-score, precision, recall and mAP) were then averaged. This procedure is used to limit the impact of training variability. Indeed, even if all the hyperparameters are set to the same values, the training algorithm leads to variability in the solutions and thus in the performance measures. The averaged performance measures are reported in a table for the validation and test sets. As the hyperparameters are optimised on the validation set, only the values obtained on this set are analysed. The ones on the test set are displayed for information.

Learning rate

As announced in previous section, following values of the learning rate were tested: 0.0005, 0.001 (default value), 0.0025 and 0.005. The table below contains the averaged performance measures for each of these values.

Learning rate	**Validation set**				**Test set**			
	F1-score	*precision*	*recall*	*mAP*	*F1-score*	*precision*	*recall*	*mAP*
0.0005	**0.792**	0.882	**0.720**	**0.607**	**0.765**	0.925	**0.652**	0.550
0.001	0.780	0.902	0.690	0.606	0.745	**0.942**	0.620	**0.562**
0.0025	0.750	**0.905**	0.645	0.576	0.712	0.935	0.577	0.523
0.005	0.737	0.887	0.635	0.602	0.702	0.935	0.567	0.540

On the validation set, a learning rate set to 0.0005 leads to slightly better results for the F1-score and the mAP than the other learning rate values. This hyperparameter is then fixed to 0.0005.

According to theory, a small learning rate should lead to better performance with a small batch size. Selecting a small learning rate is thus consistent with the theory if we consider that a batch size of 32 is relatively small.

Momentum

The values 0.7, 0.8, 0.9 (default value) and 0.95 were tested for momentum. The table below contains the averaged performance measures for each of these values.

Momentum	**Validation set**				**Test set**			
	F1-score	*precision*	*recall*	*mAP*	*F1-score*	*precision*	*recall*	*mAP*
0.7	0.777	0.870	0.695	0.591	0.752	0.920	0.637	0.527
0.8	0.787	**0.895**	0.705	0.604	0.757	0.917	0.647	0.539
0.9	**0.792**	0.882	**0.720**	**0.607**	0.765	0.925	0.652	**0.550**
0.95	0.780	0.867	0.707	0.594	**0.770**	**0.930**	**0.657**	0.546

On the validation set, the F1-score and the mAP of the momentum set to 0.9 are slightly better than the other values. This hyperparameter is then fixed to 0.9, its default value.

In theory, a large momentum term implies faster convergence and it should be larger than the learning rate. In practice, no observation was made regarding the speed of convergence and the momentum set at 0.9 is indeed larger than a learning rate set to 0.0005.

Decay

Following values were tested for the decay: 0.00005, 0.0005 (default value), 0.005 and 0.05. The table below contains the averaged performance measures for each of these values.

Decay	**Validation set**				**Test set**			
	F1-score	*precision*	*recall*	*mAP*	*F1-score*	*precision*	*recall*	*mAP*
0.00005	0.770	**0.902**	0.672	0.590	0.730	0.930	0.605	0.532
0.0005	0.792	0.882	**0.720**	**0.607**	**0.765**	0.925	**0.652**	0.550
0.005	**0.797**	0.895	**0.720**	0.601	0.755	0.912	0.645	0.553
0.05	0.770	0.890	0.680	0.600	0.742	**0.940**	0.615	**0.555**

On the validation set, the F1-score is the best when the decay is set to 0.005, but the mAP is the best when it is set to 0.0005. Whatever the value of the decay, the F1-score is better than the mAP, which is not very good, so it is preferable to increase the mAP as much as possible. Thus, the decay is set to 0.0005.

According to theory, a small decay increases the risk of overfitting, leads to a more powerful model and is more appropriate for complex datasets. The results seem consistent with theory. Indeed, a small decay has been selected because it leads to better performance measures and the dataset used for training is quite complex since it consists of small ships, which are difficult to detect.

Steps-Scales

The values tested for steps are 6000-6750 and 5000-6500, and those for scales are 0.1-0.1 and 0.5-0.2. The four combinations of values were tested. The table below contains the averaged performance measures for each steps-scales value mentioned above.

Steps	**Scales**	**Validation set**				**Test set**			
		F1-score	*precision*	*recall*	*mAP*	*F1-score*	*precision*	*recall*	*mAP*
6000-6750	**0.1-0.1**	0.792	0.882	0.720	**0.607**	0.765	0.925	0.652	**0.550**
6000-6750	0.5-0.2	0.790	0.890	0.712	0.606	0.762	0.920	0.650	0.548
5000-6500	0.1-0.1	0.785	**0.912**	0.692	0.597	0.745	**0.932**	0.625	0.548
5000-6500	0.5-0.2	**0.797**	0.867	**0.742**	0.586	**0.775**	0.902	**0.675**	0.538

On the validation set, the F1-score is the best when steps-scales is set to 5000-6500 with 0.5-0.2, but the mAP is the best when it is set to 6000-6750 with 0.1-0.1. The F1-score is respectively equal to 0.797 and 0.792, while the mAP is respectively equal to 0.586 and 0.607. The difference between the values obtained for the F1-score is very small, whereas that between the values obtained for the mAP is greater. Moreover, as the mAP is poor compared to the F1-score, it is better to keep it higher by setting steps-scales to 6000-6750 with 0.1-0.1, which is its default value.

Summary

The following table summarises which value has been selected for each hyperparameter of YOLOv3.

Hyperparameter	**Value**
Image size	608×608
Batch size	32
Subdivisions	16
Learning rate	0.0005
Burn in	1000
Momentum	0.9
Decay	0.0005
Steps-Scales	6000-6750 with 0.1-0.1

This final model has following performance measures on the validation and test sets of P dataset.

Validation set				**Test set**			
F1-score	*precision*	*recall*	*mAP*	*F1-score*	*precision*	*recall*	*mAP*
0.792	0.882	0.720	0.607	0.765	0.925	0.652	0.550

As the model has been optimised, the performance measures obtained on the test set can now be reported. The F1-score is 0.765 and the mAP is 0.550, which is poor.

4.2.3 Assessment and difficulties of the optimised model

This section presents the assessment of the optimised model on the four datasets (P, PN, P2N and the complete one), firstly in terms of Darknet performance measures, than using external ones. Afterwards, the visual difficulties of the optimised model will be explained.

4.2.3.1 Darknet performance measures

In the preceding, the final model of YOLOv3 has been tuned on P dataset and the performance measures have been obtained on the validation and test sets of P dataset. This final model was trained independently on the training sets of PN, P2N and complete datasets and the performance measures are then reported on their validation and test sets.

Datasets	**Validation set**				**Test set**			
	F1-score	*precision*	*recall*	*mAP*	*F1-score*	*precision*	*recall*	*mAP*
P	**0.792**	0.882	**0.720**	0.607	**0.765**	**0.925**	**0.652**	**0.550**
PN	0.747	0.895	0.642	0.572	0.675	0.837	0.565	0.528
P2N	0.722	**0.910**	0.605	0.555	0.692	0.897	0.572	0.546
Complete	0.782	0.870	0.715	**0.659**	0.687	0.812	0.600	0.508

Here, the performance measures obtained on the test sets are compared for model assessment. Those obtained on the validation sets are displayed as information since they were used only for model optimisation.

When comparing the performance measures of the test sets, the F1-score and the mAP obtained on that of the P dataset are the best. The mAP is slightly better than those on the other datasets, while the difference between the F1-scores is larger. Moreover, the precision and the recall are better on the test set of P dataset, which indicates less false positive and less false negatives. The number of false positives therefore appears to be higher in datasets containing negative images than in the dataset containing only positive images.

4.2.3.2 External performance measures

The table below contains the same information as the previous table, but this time using the external performance measures rather than those of Darknet.

Datasets	**Validation set**				**Test set**			
	F1-score	*precision*	*recall*	*mAP*	*F1-score*	*precision*	*recall*	*mAP*
P	0.681	0.758	0.617	0.536	**0.647**	**0.782**	**0.552**	**0.480**
PN	0.638	**0.766**	0.549	0.482	0.582	0.725	0.488	0.424
P2N	0.602	0.761	0.506	0.435	0.583	0.757	0.483	0.411
Complete	**0.688**	0.765	**0.629**	**0.555**	0.571	0.676	0.496	0.425

The same conclusions as those reached with Darknet performance measures are obtained: the F1-score and the mAP are the best on the test set of P dataset. Moreover, the number of false positive and false negative is lower on that set. In general, the external performance measures are poorer than those of Darknet, which seems to overestimate them. This seems logical, since the latter are computed using the confidence threshold that maximises them. Here, the mAP is really poor since it does not even reach 0.5.

Furthermore, regardless of the method used to compute the performances, we obtain better results for the F1-score and mAP on the complete dataset than those obtained by the creators of LS-SSDD-v1.0, who achieved an F1-score of 0.40 and mAP of 0.246.

4.2.3.3 Visual difficulties

In order to analyse visual problems and difficulties of the model, true positive (TP), false positive (FP) and false negative (FN) bounding boxes are displayed on the images of the test dataset.

True positives are displayed in green. These correspond to predictions correctly identified by the model. An image with 4 true positive bounding boxes is illustrated in Figure 4.3. False positive bounding boxes, which are wrongly detected predictions, are displayed in red, while false negative bounding boxes, which are ships not detected by the model, are displayed in blue. Figure 4.4 illustrates this. There are one false positive bounding box on the left image and two false negative bounding boxes on that on the right.

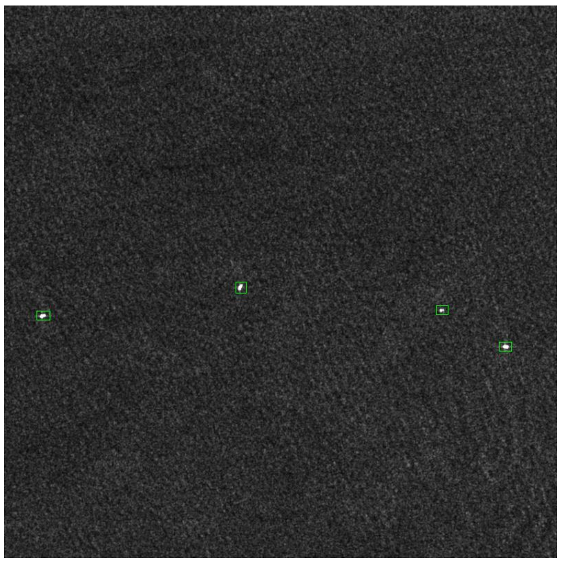


Figure 4.3: Image 01_10_27.jpg of the LS-SSDD-v1.0 dataset: correctly detected bounding boxes displayed in green.

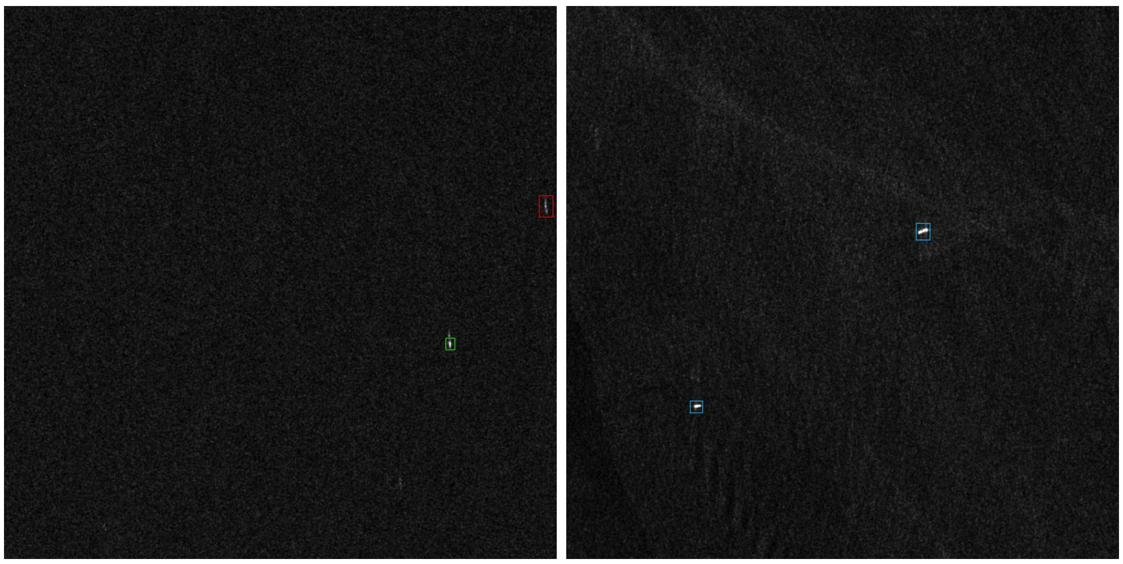


Figure 4.4: Images 12_13_4.jpg (left) and 01_10_12.jpg (right) of the LS-SSDD-v1.0 dataset: wrongly detected bounding boxes displayed in red and missed ships displayed in blue.

Several difficulties have been observed.

- The most frequent problem is that a detection is considered false (FP), while it seems visually correct, because the ground truth bounding box and the generated box have an IoU smaller than 0.5. In other words, the algorithm correctly detects the presence of the ship, but this detection is misclassified because of the lack of overlap. This problem is detected because there are two bounding boxes around the same ship: one blue (FN) and one red (FP). This should be a single bounding box which is green (TP). For this purpose, a smaller threshold on IoU should be used. Figure 4.5 illustrates this problem.

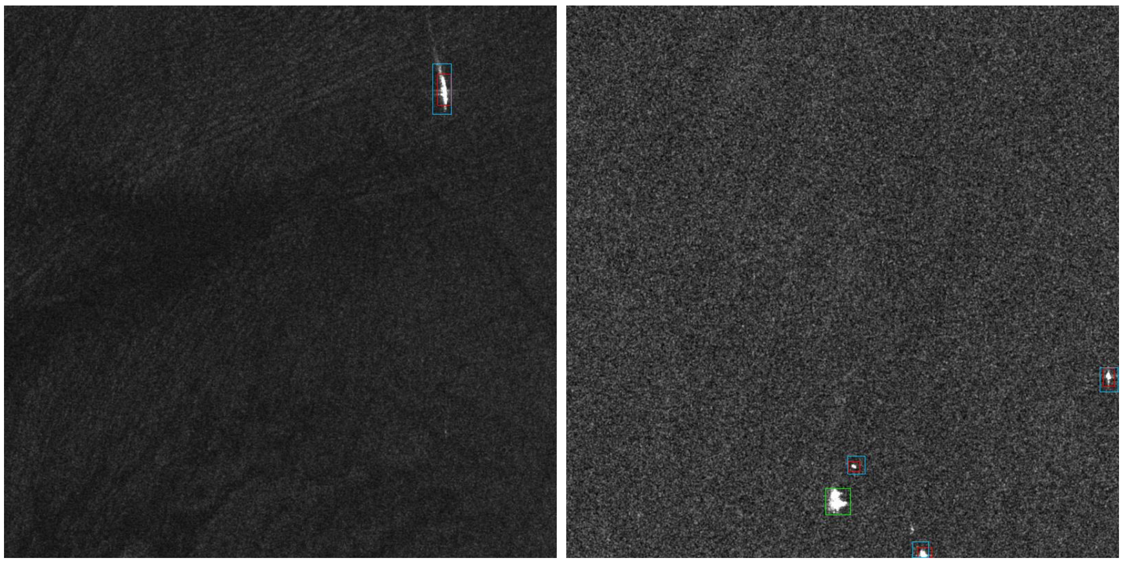


Figure 4.5: Images 02_19_23.jpg (left) and 05_10_17.jpg (right).

- Many normal sized ships in open sea (hence without detection difficulties) are not detected although visually they should be detected easily. A possible explanation for this problem could be that the model confuses these ships with islands. This is illustrated in Figure 4.6. One solution could be to feed the algorithm with several images of the same scene taken at different times so that it can learn to distinguish between static and moving features.

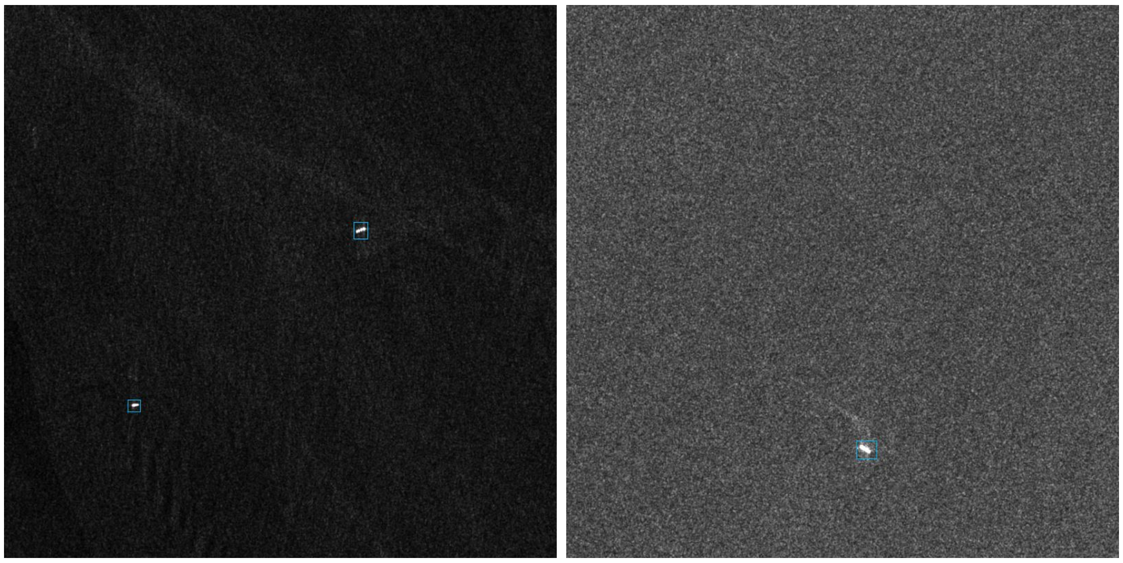


Figure 4.6: Images 01_10_12.jpg (left) and 09_3_17.jpg (right).

- Many ships are not detected near coasts or in harbours. In these situations, there are few false positives, but many false negatives and some good detections. So, YOLOv3 seems to abstain a lot when it should detect near coasts or in harbours. Figures 4.7 and 4.8 illustrate this problem.
- Tiny ships are not correctly detected, as illustrated in Figure 4.9. One possible reason could be that the small ships are built with different materials, probably wood, which appears less bright on SAR images.

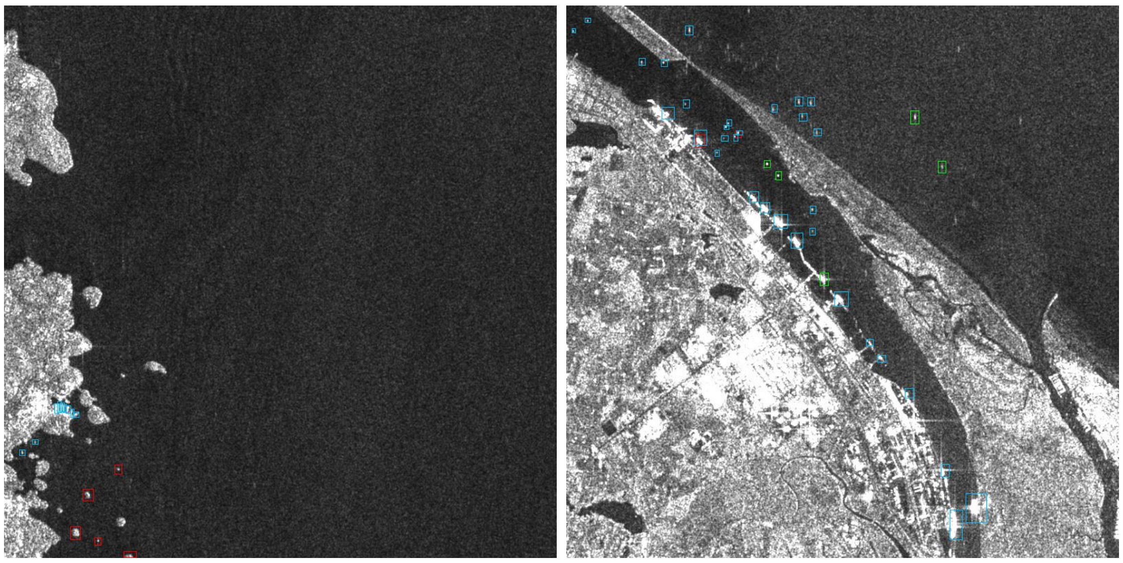


Figure 4.7: Images 02_18_17.jpg (left) and 09_18_14.jpg (right).

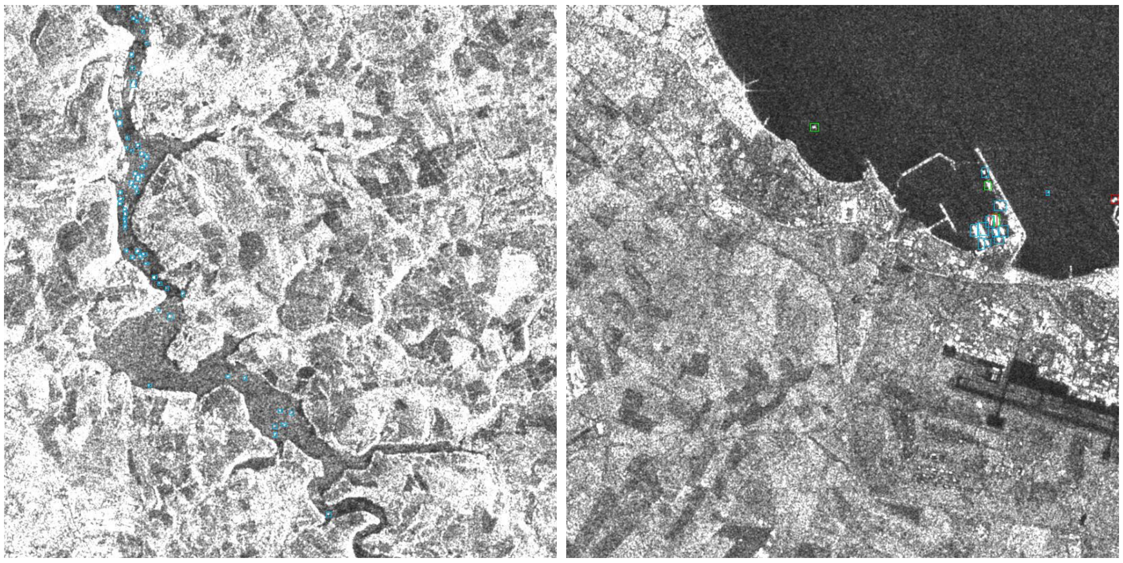


Figure 4.8: Images 10_18_28.jpg (left) and 09_10_9.jpg (right).

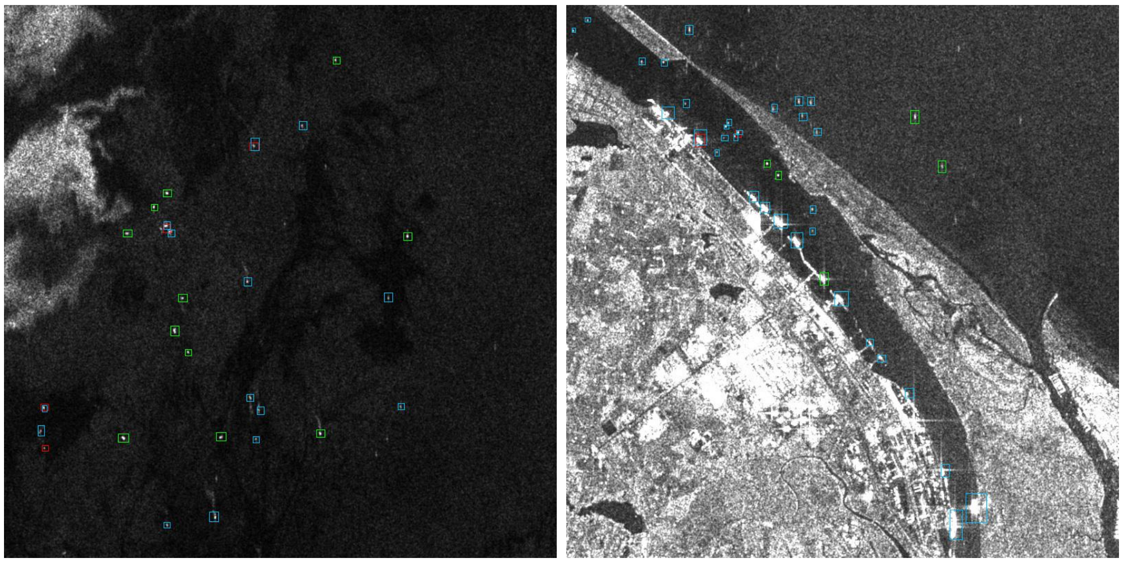


Figure 4.9: Images 08_14_13.jpg (left) and 09_18_14.jpg (right).

- Small bright dots in the sea are identified as ships when they possibly are small pieces of land (FP). This is illustrated in Figure 4.10, where the small dots detected as ships in the images are indeed pieces of land.

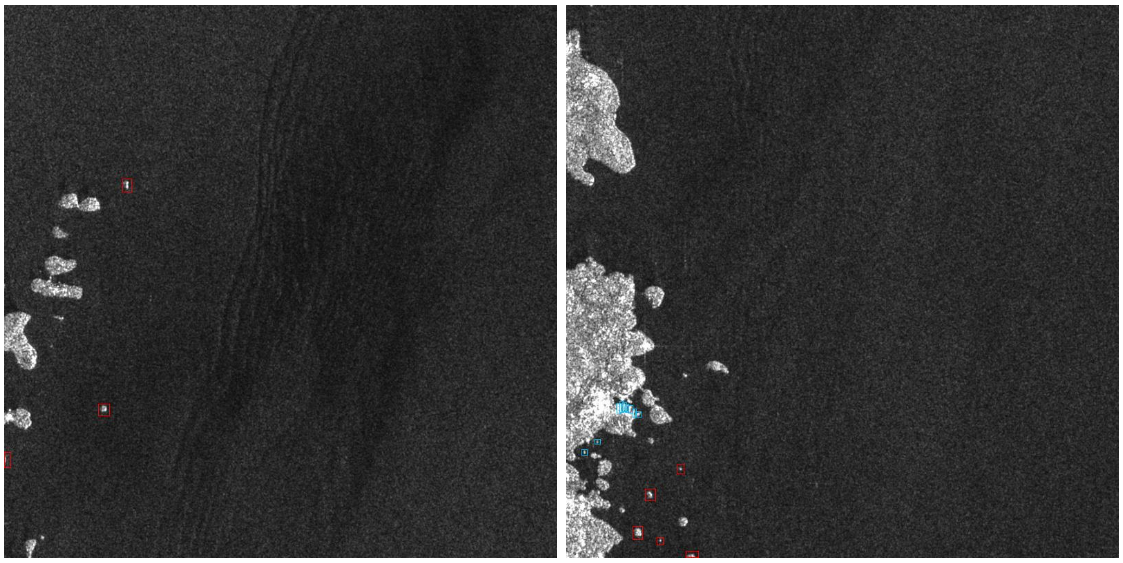


Figure 4.10: Images 02_17_17.jpg (left) and 02_18_17.jpg (right).

- Ships are not correctly detected when their contrast with the background is low (FN). This is illustrated in Figure 4.11.

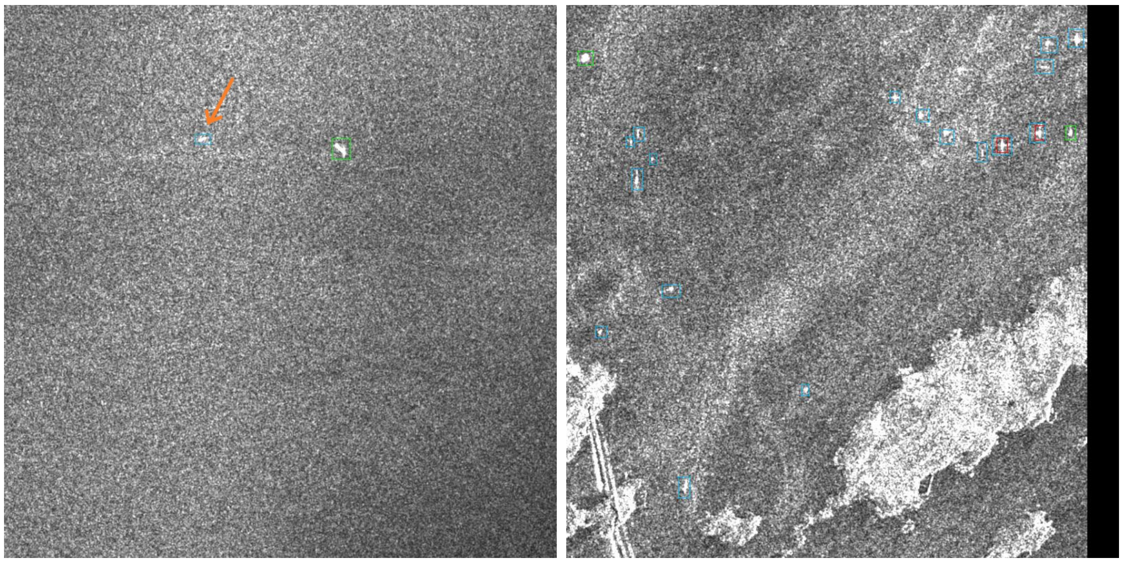


Figure 4.11: Images 15_15_19.jpg (left) and 11_13_30.jpg (right).

- Some false negatives and false positives appear on the edge of some images: these poor detections are probably due to the fact that the ship is cut in two on the border of the image. This implies that the neural network misses the context of the object. Modifying the padding parameter in convolutional layers could contribute to the problem.

  In the case of false negatives, the reason may be that the algorithm does not recognise that part of the ship, which is too small and incomplete. This is illustrated in Figure 4.12.

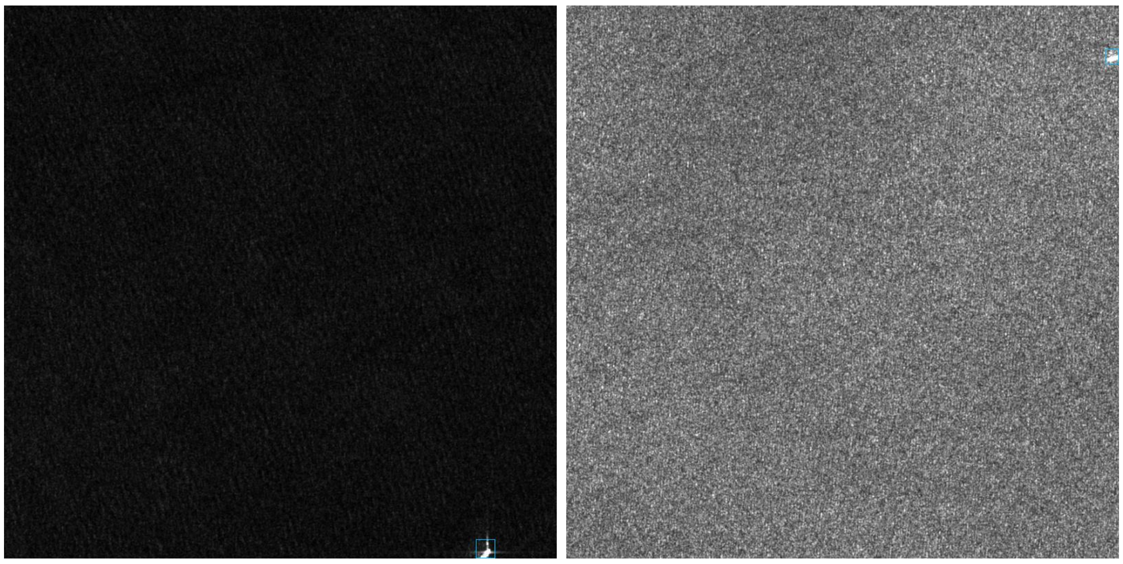


Figure 4.12: Images 01_7_6.jpg (left) and 06_8_14.jpg (right).

The problem with false positives is illustrated in Figure 4.13 on images 01_14_18.jpg and 02_17_21.jpg (bottom left and bottom right) with false positive bounding boxes at their top edges. In this case, the reason is that the ground truth bounding box has been removed on the corresponding image due to the cutting and it only appears on the image containing the other part of the ship. This is illustrated on left images in Figure 4.13. The bottom image, 01_14_18.jpg, which has a false positive detection on its top edge, was side by side with image 01_13_18.jpg in the large-scale image. There is one ship on the cut between these images, the largest part of this ship is on the top image, while the bottom image contains only a small part of it. From this, the ground truth bounding box of this ship appears only on the top image, whereas there is none on the bottom image. Thus, this detection is classified as FP because there is no ground truth available on the bottom image, but this detection is correct and should be classified as TP.

Similarly, on the right in Figure 4.13, images 02_16_21.jpg and 02_17_21.jpg were side by side in the large-scale image and, because of the cutting, a ship was cut into two parts, whose ground truth bounding box is on the top image and not on the bottom, implying a false positive, whereas it is a correct detection.

To overcome this problem, when a ship is cut in two and separated on two images, a ground truth bounding box should be kept for each part and not just for the larger one.

- A few false positives identify a ship on land, for instance in mountains, farmlands or urban areas. This is illustrated in Figure 4.14 and 4.15. This problem could be solved by using a land-sea mask indicating where areas are land or sea. Applying such a mask would permit to remove detections on land.

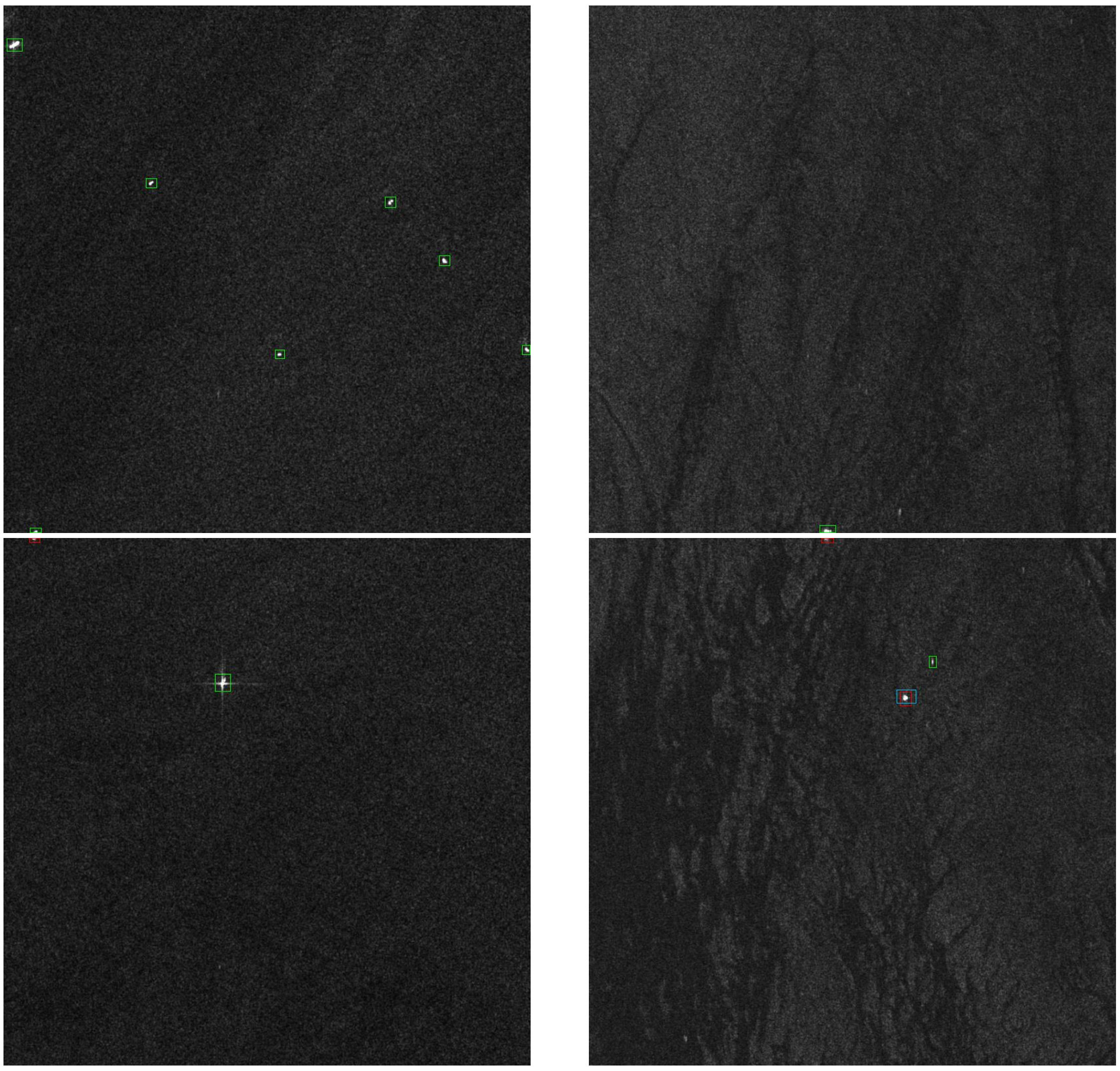


Figure 4.13: Images 01_13_18.jpg (top left), 01_14_18.jpg (bottom left), 02_16_21.jpg (top right) and 02_17_21.jpg (bottom right).

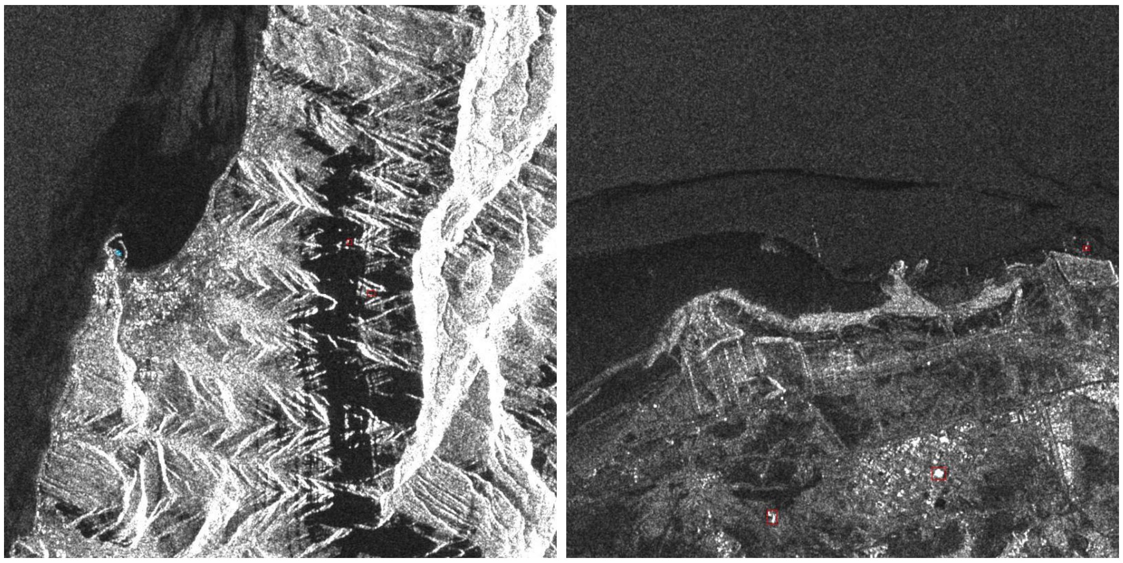


Figure 4.14: Images 04_13_24.jpg (left) and 04_2_22.jpg (right).

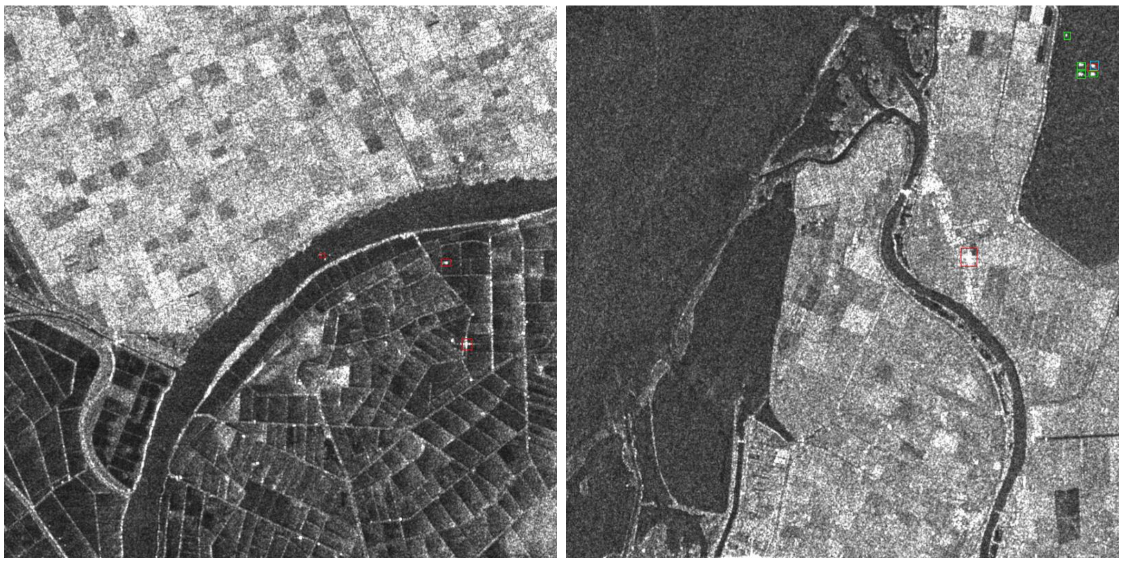


Figure 4.15: Images 09_15_6.jpg (left) and 02_17_27.jpg (right).

To summarise, the visual difficulties of YOLOv3 are listed below.

- The most frequent problem is that a detection is considered false, while it seems visually correct, because the ground truth bounding box and the generated box have an IoU smaller than 0.5. In other words, the algorithm correctly detects the presence of the ship, but this detection is misclassified because of the lack of overlap.
- Many normal sized ships in the open sea (hence without detection difficulties) are not detected.
- A lot of ships are not detected near coasts or in harbours. In these situations, there are few false positives, but many false negatives or some good detections.
- Tiny ships are difficult to detect.
- Small bright dots in the sea are identified as ships when they possibly are small pieces of land.
- Ships are not correctly identified when their contrast with the background is low.
- Some false negatives and false positives appear on the edge of some images: these poor detections are probably due to the fact that the ship is cut in two on the border of the image.
  - For false negatives, this may be due to the fact that the algorithm does not recognise that part of the ship, which is too small and incomplete.
  - For false positives, the ground truth bounding box has been removed on the corresponding image because of the cutting and it only appears on the image containing the other part of the ship.
- Parts of the land are misidentified as ships.

4.2.4 Threshold optimisation: reducing some difficulties

In the previous section, the threshold hyperparameters were set to their default values, i.e. 0.5 for the intersection over union (IoU) and 0.3 for the confidence threshold. As several difficulties were encountered, we now modify them to reduce some difficulties and thus increase the performance measures. We first describe the analysis of the thresholds modification. This study is based on the influence on visual difficulties and on the behaviour of performance measures with respect to thresholds. Then, we present the effects on the performance measures and on the visual difficulties.

As explained at the beginning of this chapter, the analyses in this section are based solely on PN dataset.

4.2.4.1 Threshold modification

The intersection over union (IoU) threshold is used to classify a detection as true if it has an IoU with a ground truth bounding box larger than the threshold and to classify it as false otherwise. The confidence threshold is used to keep only the predictions whose confidence score is larger than this threshold. By default, these are set to 0.5 for the IoU and 0.3 for the confidence. The new thresholds are respectively set to 0.3 and 0.2. The reasoning behind this choice is explained below.

When studying the IoU threshold, it has been observed that the performance measures (mAP and F1-score) increase when the IoU threshold decreases. However, it should not be too small, because it is necessary to correctly identify the generated box with the right ground truth bounding box. For this purpose, these boxes need an IoU which is not too small. Moreover, this threshold should be taken smaller than 0.5 (default threshold) to avoid as much as possible the problem of false positive and false negative of a same element, i.e. the visual problem discussed previously. So, different IoU threshold values have been tested. With an IoU threshold of 0.4, this problem still occurs too frequently, while it occurs rarely with an IoU threshold set to 0.3. Furthermore, this seems a sufficient IoU to map a predicted bounding box with the right ground truth box. With an IoU threshold of 0.1, this problem almost never occurs, but it is a too small IoU for the mapping. Hence, with an IoU threshold of 0.3, a lot of false negatives and false positives are replaced by true positives, which significantly decreases the number of false negatives and false positives and significantly increases the number of true positives.

When studying the confidence threshold, it has been observed that there is a small increase of the performance measures (mAP and F1-score) when taking a confidence of 0.2 instead of 0.3. Indeed, more predictions are considered and most of them are true positives, which significantly increases their number.

Thus, in the following, the IoU and confidence thresholds are respectively set to 0.3 and 0.2.

4.2.4.2 Performance measures

The performance measures[4] of YOLOv3's final model on PN dataset when using the new and default thresholds are given in the table below. Modifying the thresholds significantly increased the F1-score, the recall and the mAP, while the precision has only slightly increased.

Thresholds		**Validation set**				**Test set**			
IoU	*Confidence*	*F1-score*	*precision*	*recall*	*mAP*	*F1-score*	*precision*	*recall*	*mAP*
0.3	**0.2**	**0.747**	**0.801**	**0.702**	**0.663**	**0.686**	**0.738**	**0.644**	**0.587**
0.5	0.3	0.638	0.766	0.549	0.482	0.582	0.725	0.488	0.424

[4]Here, the external performance measures are used, not those of Darknet.

4.2.4.3 Visual difficulties

The modification of the thresholds settings imply changes in visual difficulties.

- The visual problem of false negative and false positive of a same ship occurs rarely, these are now considered as true positives. In order to illustrate this change, images from Figure 4.5, where the default thresholds were used, are now displayed in Figure 4.16 when using their new thresholds.

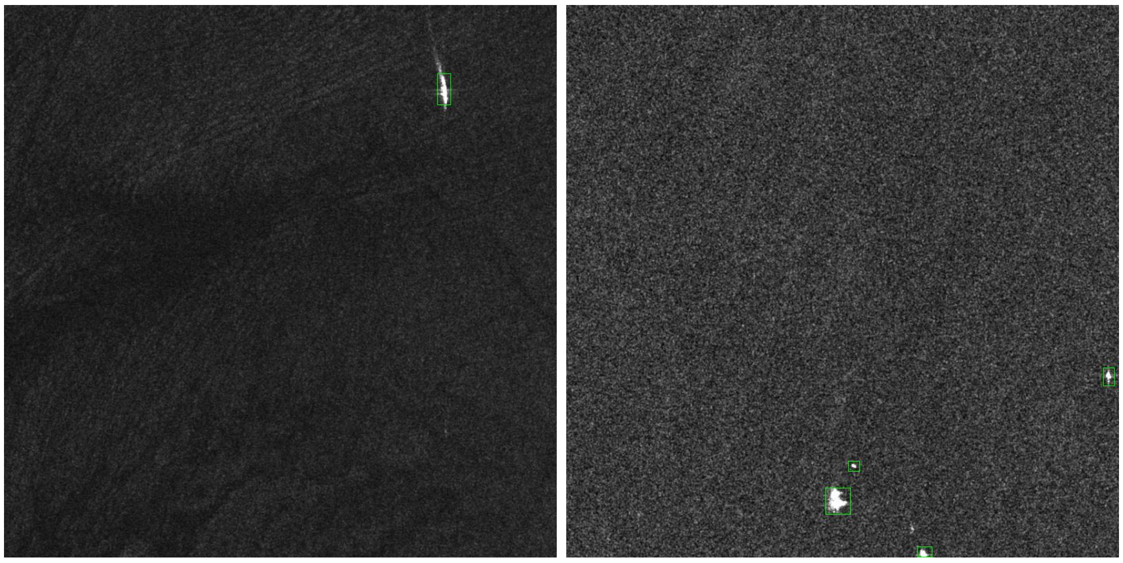


Figure 4.16: Images 02_19_23.jpg (left) and 05_10_17.jpg (right).

  Hence, taking a smaller IoU threshold has decreased the number of false negatives and false positives and has then increased the number of true positives.

- In general, the change of the confidence threshold also implies fewer false negatives, but more false positives and true positives. Indeed, a smaller confidence threshold increases the number of detections, some of them are true positives, while the other are false positives. This decreases the number of false negative.

  There are less false negatives and more true positives. The images from Figure 4.6, where the default thresholds were used, are displayed in Figure 4.17 using their new thresholds. These ships, previously not detected, are now correctly detected, which increases the number of true positives.

  The change in the confidence threshold has resulted in more false positives being detected, especially on land. For instance, the images from Figure 4.14, where default thresholds were used, are now displayed in Figure 4.18, using their new thresholds. This illustrates the impact of the change of thresholds. In this case, there are more false positives detected on lands.

- The other difficulties are the same as those discussed previously.

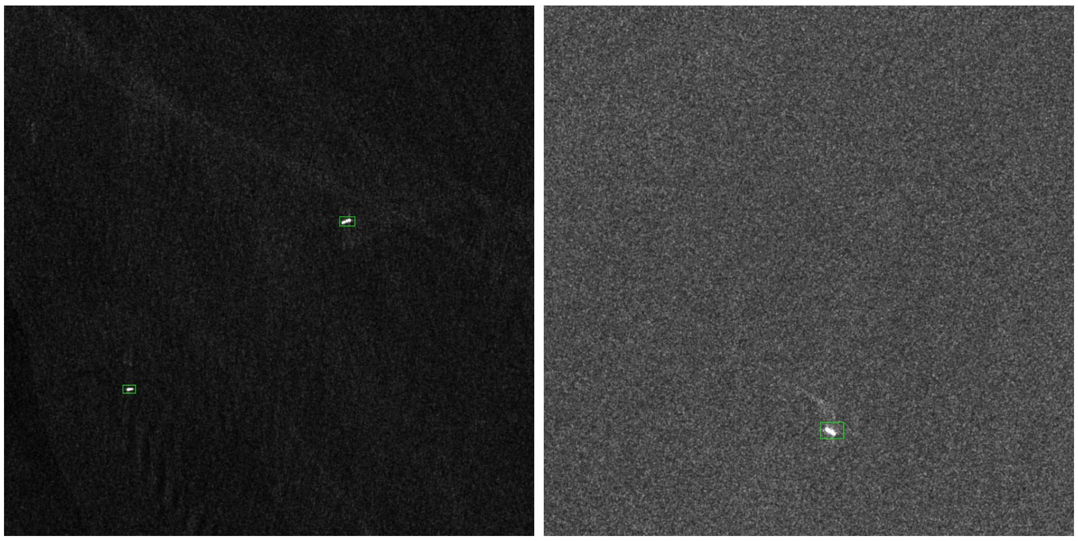


Figure 4.17: Images 01_10_12.jpg (left) and 09_3_17.jpg (right).

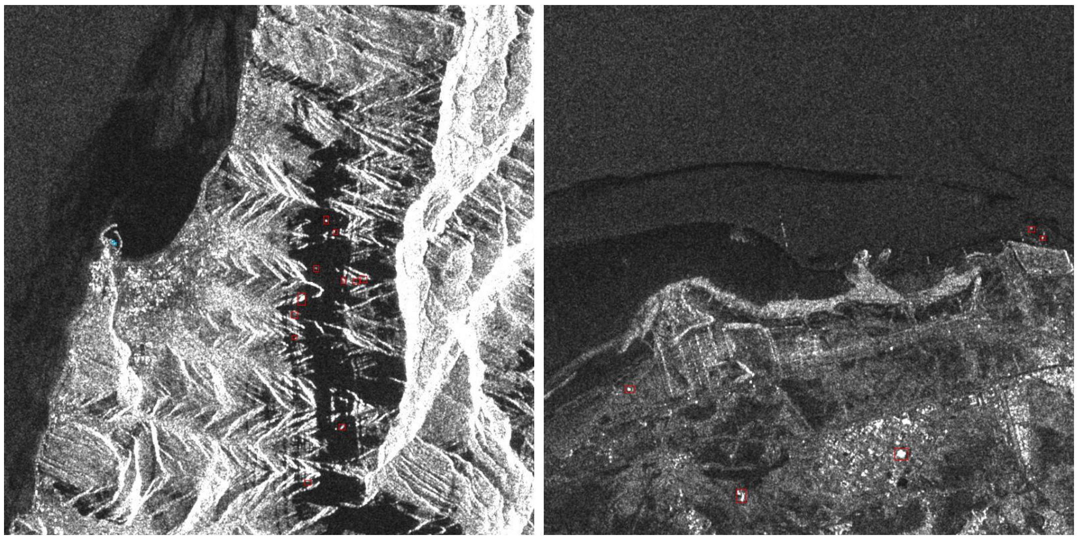


Figure 4.18: Images 04_13_24.jpg (left) and 04_2_22.jpg (right).

By modifying the thresholds, some difficulties were solved. To summarise, the remaining difficulties are listed below.

- Some normal sized ships in open sea (hence without detection difficulties) are still not detected.
- A lot of ships are not detected near coasts or in harbours. There are few false positives and some good detections, but many false negatives.
- Tiny ships are difficult to detect.
- Small bright dots in the sea are identified as ships when they possibly are small pieces of land.
- Ships are not correctly identified when their contrast with the background is low.
- Some false negatives and false positives appear on the edge of some images, these poor detections are probably due to the fact that the ship is cut in two on the border of the image.
  - For false negatives, this may be due to the fact that the algorithm does not recognise that part of the ship, which is too small and incomplete.

  - For false positives, the ground truth bounding box has been removed on the corresponding image because of the cutting and it only appears on the image containing the other part of the ship.
- Parts of the land are misidentified as ships.

4.3 YOLO version 4

This section first presents the optimisation of YOLOv4 hyperparameters, then the assessment of the optimised model and finally, its visual difficulties compared to those of YOLOv3. As the process is very similar to YOLOv3 optimisation, far fewer details are provided.

4.3.1 Hyperparameters optimisation

The hyperparameters used in YOLOv4 are the same as those of YOLOv3, see Section 4.2.1, and are set to the same initial values, except the momentum hyperparameter that is set to 0.949 instead of 0.9.

As for YOLOv3, the hyperparameters optimisation is split into two parts, one for the global hyperparameters and one for the local ones. The same behaviour of the global hyperparameters has been assumed because the neural network backbone in these two versions is the same and the difference between them lies in the addition of the "Bag of Freebies" and "Bag of Specials" methods to improve performance, see the YOLO history in 2.4.2.

We remind here the conclusions from Section 4.2.2.1 about the optimisation of YOLOv3: an image size of 608×608, the largest possible batch size, the smallest possible subdivision value, the burn in set to its default value and the learning rate within the defined range of values. With an image of this size, the batch size must be equal to subdivision, otherwise learning would crash due to too high memory consumption. The model with a batch size and subdivision equal to 32 was selected because it led to the best results with a not too long training time (about 8:30 hours). In YOLOv3, some models with equal batch size and subdivision led to instabilities during training, but this problem is not encountered with this model.

The local hyperparameter study is not presented here but in Appendix C since the development is very similar to that of YOLOv3. The following table lists the value selected for each hyperparameter.

Hyperparameter	**Value**
Image size	608×608
Batch size	32
Subdivisions	32
Learning rate	0.0025
Burn in	1000
Momentum	0.949
Decay	0.0005
Steps-Scales	6000-6750 with 0.1-0.1

This final model has following performance measures on the validation and test sets of P dataset.

Validation set				**Test set**			
F1-score	*precision*	*recall*	*mAP*	*F1-score*	*precision*	*recall*	*mAP*
0.842	0.867	0.820	0.690	0.782	0.880	0.705	0.606

As the model has been optimised, the performance measures obtained on the test set can be reported. The F1-score is 0.782 and the mAP is 0.606, which is not very good.

4.3.2 Assessment and difficulties of the optimised model

This section presents the assessment of the optimised model on the four datasets (P, PN, P2N and the complete one) using the external performance measures (and not those of Darknet). Afterwards, the visual difficulties of the optimised model will be explained.

4.3.2.1 Performance measures

In the previous section, model optimisation was performed on P dataset and the performance measures were given on the validation and test sets of P dataset. The final model is trained independently on the training sets of PN, P2N and complete datasets and the resulting performance measures on their validation and test sets are given in the table below.

Datasets	**Validation set**				**Test set**			
	F1-score	*precision*	*recall*	*mAP*	*F1-score*	*precision*	*recall*	*mAP*
P	**0.727**	0.748	**0.708**	**0.640**	0.659	0.740	0.594	0.533
PN	0.695	0.783	0.625	0.567	**0.726**	**0.804**	**0.663**	**0.608**
P2N	0.670	0.780	0.587	0.540	0.684	0.739	0.637	0.585
Complete	0.701	**0.818**	0.615	0.575	0.701	0.794	0.630	0.580

When comparing the performance measures of the test sets, the F1-score and the mAP obtained on that of PN dataset are the best, while those on P dataset are the worst. This suggests that the model has learned better to recognise the characteristics of the ships on the balanced dataset. Moreover, the precision and the recall are better on the test set of PN dataset, which indicates less false positives and less false negatives. Finally, the F1-score is acceptable, while the mAP is poor.

4.3.2.2 Visual difficulties

Unlike YOLOv3, YOLOv4 has no difficulty in detecting normal-sized ships in open sea. This is a situation in which, visually, the ships should be easily detected. Apart from that, the difficulties encountered by the two versions of YOLO are very similar, some are managed similarly by both methods, while others are managed better by YOLOv4. The difficulties handled similarly by both methods are the following ones:

- The most frequent problem is that a detection is considered as false (FP), while it is visually correct, because of the lack of overlap between the ground truth bounding box and the generated box. As with YOLOv3, to solve this problem, a smaller threshold on IoU should be used.
- Small bright dots in the sea are identified as ships when they possibly are small pieces of land (FP).
- Some false negatives and false positives appear on the edge of some images. The reason for these poor detections is the same as previously explained.

- Parts of the land are misidentified as ships.

The difficulties[5] better handled by YOLOv4 are the following ones:

- Many ships are not detected near coasts or in harbours, but more false positives and true positives seems to be present than with YOLOv3 (in the sense that it tries more to make predictions while YOLOv3 seems to "abstain"). This was illustrated in Figures 4.7 and 4.8. Figure 4.19 illustrates this problem for YOLOv4.

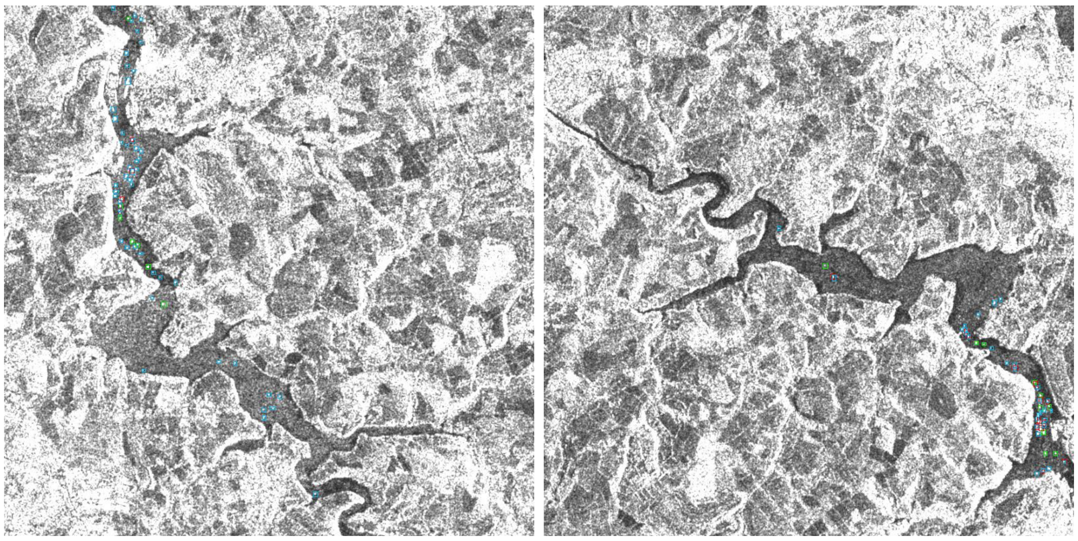


Figure 4.19: Images 10_18_28.jpg (left) and 07_2_25.jpg (right).

- Some tiny ships are not correctly detected, as illustrated in Figure 4.20.

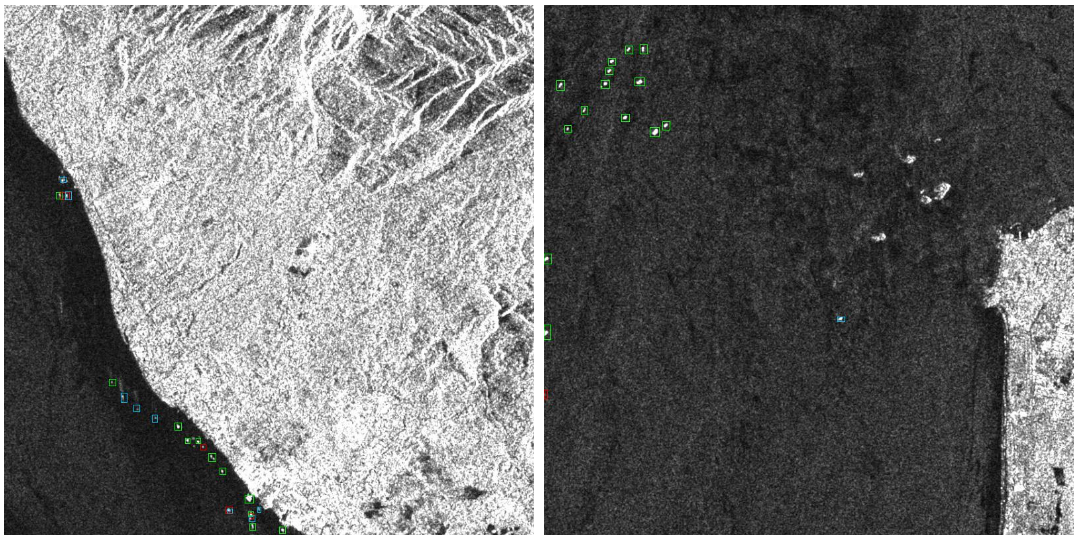


Figure 4.20: Images 08_4_21.jpg (left) and 12_6_27.jpg (right).

- Ships are not correctly identified when their contrast with the background is low. This is illustrated in Figure 4.21.

[5]As with YOLOv3, the green bounding boxes indicate true positives, the red ones false positives and the blue ones false negatives.

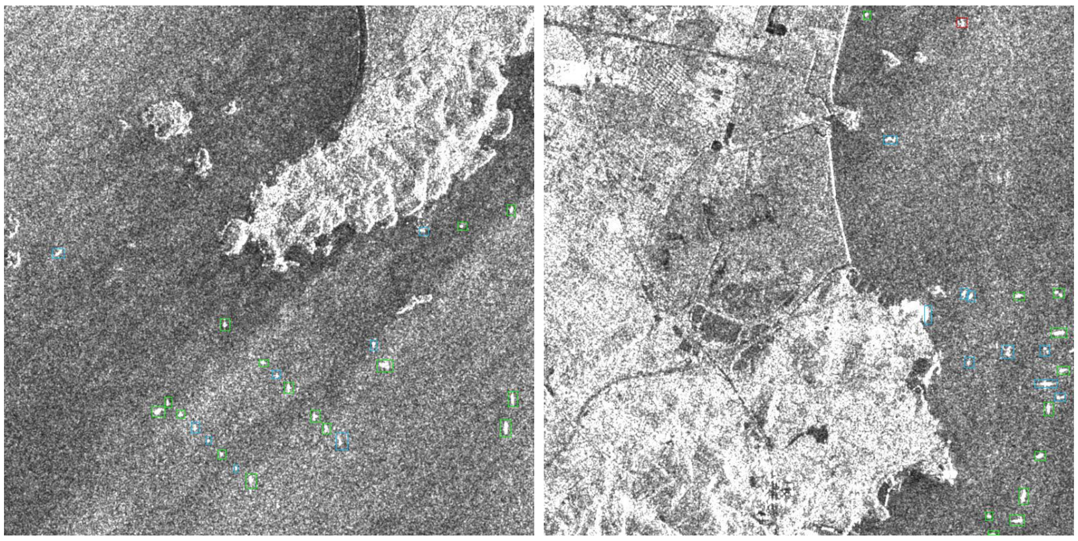


Figure 4.21: Images 11_13_28.jpg (left) and 11_12_29.jpg (right).

4.3.3 Threshold optimisation: reducing some difficulties

As with YOLOv3, the IoU and confidence thresholds can be modified to improve performance measures and reduce some visual difficulties. By default, they are set to 0.5 and 0.3, respectively. When these thresholds were analysed, the same behaviour as with YOLOv3 was observed, see Section 4.2.4.1. Thus, the IoU threshold was set at 0.3 and the confidence threshold at 0.2.

In this section, we present the effects on the (external) performance measures and on the visual difficulties. As with YOLOv3, the analysis focuses on PN dataset.

4.3.3.1 Performance measures

The performance measures of YOLOv4's final model on PN dataset when using the new and default thresholds are given in the table below. Modifying the thresholds significantly increases all performance measures, but mainly the mAP whose value increased by almost 0.15.

Thresholds		**Validation set**				**Test set**			
IoU	*Confidence*	*F1-score*	*precision*	*recall*	*mAP*	*F1-score*	*precision*	*recall*	*mAP*
0.3	**0.2**	**0.784**	**0.828**	**0.746**	**0.716**	**0.819**	**0.861**	**0.782**	**0.755**
0.5	0.3	0.695	0.783	0.625	0.567	0.726	0.804	0.663	0.608

4.3.3.2 Visual difficulties

The reduction of visual difficulties is the same as the one described for YOLOv3 in Section 4.2.4.3. The remaining difficulties are listed below.

- A lot of ships are not detected near coasts or in harbours.
- Some tiny ships are not correctly detected.
- Small bright dots in the sea are identified as ships when they possibly are small pieces of land.
- Ships are not correctly identified when their contrast with the background is low.
- Some false negatives and false positives appear on the edge of some images. The reason for these poor detections is the same as previously explained.
- Very few parts of the land are misidentified as ships.

4.4 Mask R-CNN

This section presents the Mask R-CNN hyperparameters with their default values, the optimisation of the model, its assessment and finally, the visual difficulties encountered by the optimised model compared to YOLOv4.

4.4.1 Hyperparameters

The following table contains the hyperparameters of Mask R-CNN with their default value.

Hyperparameter	**Default value**
Image size	800×800
Batch size	\
Learning rate	0.001
Momentum	0.9
Decay	0.0005
Step size - Gamma	3 - 0.1
Epoch	50

The image size was not modified during model optimisation because it was not possible to modify it in the code we were working with. The size of the original images of the LS-SSDD-v1.0 dataset was therefore used, i.e. a size of 800×800. In other words, the default size is not fixed and adapts to the size of the input images, unlike YOLO where the image is resized by default to a size of 416×416.

Because of memory limitations and because the subdivisions hyperparameter does not exist, the maximum possible batch size is 12. The purpose of the hyperparameter "Step size - Gamma" is the same as "Steps - Scales" in YOLO: for the default values, this correspond to multiply the learning rate by 0.1 every 3 epochs. Contrary to YOLO where the number of iterations must be set and the number of epochs is deduced from it, for this model, the number of epochs is set at 50 for all tests (and the number of iterations is deduced).

4.4.2 Hyperparameters optimisation

This section covers hyperparameters optimisation, with the first part focusing on global hyperparameters and the second on local ones.

The global hyperparameters study was not carried out on the same datasets as for YOLOv3: they have been studied solely on P dataset and not on PN because it would have taken too much time to do both and because with YOLOv3, the same conclusions had been reached on both datasets.

4.4.2.1 Global hyperparameters study

This section explains how we studied the influence of the global hyperparameters in order to decide how to set them. The global hyperparameters are the batch size and the learning rate. The following values were tested for them:

- **Batch size:** 4, 6, 8, 10, 12
- **Learning rate:** 0.0001, 0.0005, 0.001, 0.0025, 0.005, 0.0075

All possible combinations of values were tested. Therefore, the influence of the batch size was studied using 6 different tests, each corresponding to a different value of the learning rate. Similarly, the influence of the learning rate was studied using 5 different tests, each corresponding to a different value of the batch size. Each test was performed twice to limit the variability of the training.

Batch size

We noticed that when the batch size increases, the performance measures (F1-score, precision, recall and mAP) tend to decrease and that, according to the value of the learning rate, a batch size of 4 or 6 is preferable. As there is not much difference between the performances achieved with these sizes, it is not possible to select the best one at this stage. It will then be determined based on the behaviour of the learning rate.

Contrary to what appeared for YOLO, the mAP curve becomes stable very quickly and no longer increases at all, as shown in Figure 4.22. Moreover, the loss function does not have a decreasing behaviour.

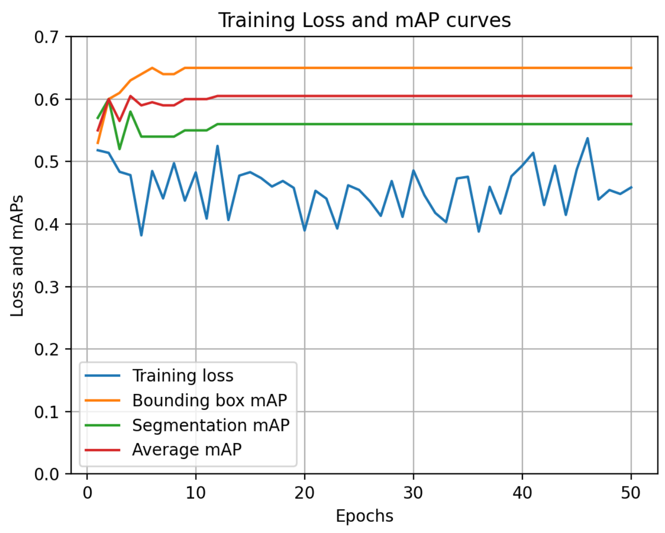


Figure 4.22: Graph of a training with a mAP curve quickly stable and a non-decreasing loss function.

The conclusions of our experiments do not reflect what is expected in theory. Indeed, theoretically, a large batch size should lead to better performance measures.

Learning rate

We noticed that when the learning rate value increases, the performance measures (F1-score, precision, recall and mAP) tend to increase and that, in most cases a learning rate set to 0.0075 performs better, but for some batch sizes (6 and 12), a learning rate of 0.0025 is better.

This is no consistent with theory since a large learning rate should lead to better performance measures with a large batch size.

Summary

To summarize, the models providing the best performance results are

Model	Batch size	Learning rate
Model 1	4	0.0075
Model 2	6	0.0025

Among these models, model 1 is selected because it leads to slightly better performance measures.

4.4.2.2 Local hyperparameters study

This section explains how we studied the influence of the local hyperparameters in order to decide how to set them. The local hyperparameters are the momentum, decay and step size - gamma. The learning rate is no longer considered as a local hyperparameter here because it depends on the batch size, which was the only global hyperparameter considered, so it could be set directly above.

As for YOLO, for each local hyperparameter, several values were tested and four tests for each of them were performed and the resulting performance measures (F1-score, precision, recall and mAP) were then averaged to limit the impact of training variability. These averaged performance measures are reported in a table for the validation and test sets. The validation set is used for the hyperparameter tuning, while the values obtained on the test set are displayed for information and thus, are not analysed.

Momentum

The values 0.7, 0.8, 0.9 (default value) and 0.95 were tested for momentum. The table below contains the averaged performance measures for each of these values.

Momentum	**Validation set**				**Test set**			
	F1-score	*precision*	*recall*	*mAP*	*F1-score*	*precision*	*recall*	*mAP*
0.7	0.544	0.440	0.712	0.637	0.583	0.468	0.773	0.681
0.8	0.532	0.417	**0.734**	**0.649**	0.564	0.444	0.773	0.688
0.9	0.570	0.469	0.729	0.641	0.613	0.505	0.780	**0.695**
0.95	**0.575**	**0.474**	0.729	0.636	**0.618**	**0.511**	**0.782**	0.691

On the validation set, the highest value for the F1-score is obtained with a momentum of 0.95 and for the mAP with a value of 0.8. As these values also lead to the worst values for the mAP and F1-score respectively, the momentum was set at 0.9 because it offers a good trade-off. According to theory, this term should be set larger than the learning rate, which is indeed the case.

Decay

Following values were tested for the decay: 0.00005, 0.0005 (default value), 0.005 and 0.05. The table below contains the averaged performance measures for each of these values.

Decay	**Validation set**				**Test set**			
	F1-score	*precision*	*recall*	*mAP*	*F1-score*	*precision*	*recall*	*mAP*
0.00005	0.571	0.462	0.747	0.656	0.614	0.507	0.778	0.693
0.0005	0.570	**0.469**	0.729	0.641	0.613	0.505	0.780	0.695
0.005	**0.577**	0.468	**0.753**	**0.661**	**0.621**	**0.516**	0.781	**0.696**
0.05	0.572	0.462	0.750	0.655	0.618	0.511	**0.782**	0.695

The decay value is set to 0.005 because it leads to the best F1-score and mAP on the validation set. This is not really consistent with theory because a small decay is supposed to lead to better performance measures and to be more appropriate with a complex dataset, which is our case.

Step size - Gamma

The values tested for the step size are 3 and 5, and those for gamma are 0.1 and 0.5. The four combinations of values were tested. Recall that the default value is a step size of 3 and gamma set to 0.1. The table below contains the averaged performance measures for each step size - gamma value mentioned above.

Steps	**Scales**	**Validation set**				**Test set**			
		F1-score	*precision*	*recall*	*mAP*	*F1-score*	*precision*	*recall*	*mAP*
3	0.1	0.577	0.468	0.753	0.661	0.621	0.516	0.781	0.696
3	0.5	0.625	0.506	0.812	0.741	0.648	0.561	0.766	0.691
5	0.1	0.587	0.467	0.789	0.702	0.622	0.531	**0.792**	**0.709**
5	**0.5**	**0.695**	**0.598**	**0.829**	**0.769**	**0.685**	**0.638**	0.740	0.673

Modifying this hyperparameter significantly improved the performance measures on the validation set. A step size - gamma set to 5 - 0.5 leads to the best results. In addition, during training, the mAP curve varies more and the loss function behaves better because it tends to decrease, allowing the model to learn better, see Figure 4.23.

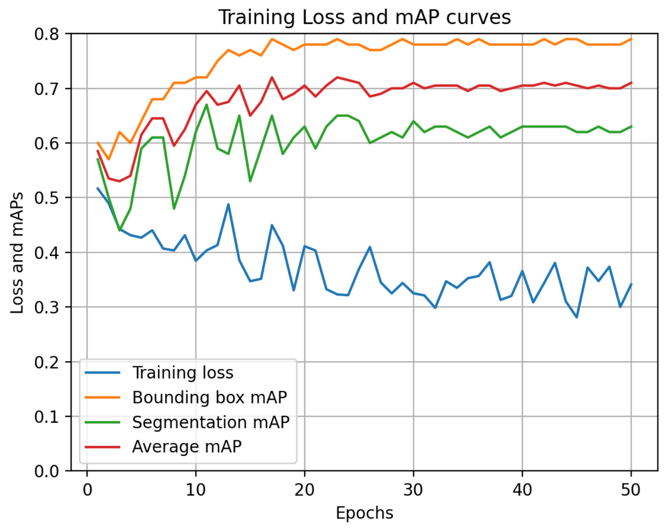


Figure 4.23: Graph of a training with its mAP curves and a decreasing loss function.

Summary

The following table summarises which value has been selected for each hyperparameter of Mask R-CNN.

Hyperparameter	**Value**
Image size	800 × 800
Batch size	4
Learning rate	0.0075
Momentum	0.9
Decay	0.005
Steps-Scales	5 with 0.5

This final model has following performance measures on the validation and test sets of P dataset.

Validation set				**Test set**			
F1-score	*precision*	*recall*	*mAP*	*F1-score*	*precision*	*recall*	*mAP*
0.695	0.598	0.829	0.769	0.685	0.638	0.73975	0.673

As the model has been optimised, the performance measures obtained on the test set can now be reported: the F1-score is 0.685 and the mAP is 0.673.

4.4.3 Assessment and difficulties of the optimised model

This section presents the assessment of the optimised model on the four datasets (P, PN, P2N and the complete one) using the external performance measures. Afterwards, the visual difficulties of the optimised model will be explained.

4.4.3.1 Performance measures

In the preceding, the final model of Mask R-CNN has been tuned on P dataset and the performance measures have been obtained on the validation and test sets of P dataset. This final model is trained independently on the training sets of PN, P2N and complete datasets and the performance measures are then given on the validation and test sets of these other datasets.

Datasets	**Validation set**				**Test set**			
	F1-score	*precision*	*recall*	*mAP*	*F1-score*	*precision*	*recall*	*mAP*
P	0,695	0,598	0,829	0,769	0,685	0,638	**0,740**	0,673
PN	**0,718**	**0,629**	**0,836**	**0,799**	**0,698**	**0,656**	0,731	**0,714**
P2N	0,692	0,597	0,821	0,761	0,601	0,580	0,624	0,573
Complete	\							

The training was not possible on the complete dataset. With the Mask R-CNN implementation used, it was not possible to split the dataset into training/validation/test subsets, probably due to the dataset being too large, so training could not be tried.

When comparing the performance measures of the test sets, the F1-score and the mAP obtained on that of PN dataset are the best, the ones obtained on P dataset are very similar, while those on P2N dataset are the worst. This suggests that the model learns better the characteristics of the ships on the P and PN datasets. Moreover, the precision and the recall are better on the test sets of P and PN, indicating less false positives and less false negatives.

Recall that the authors of LS-SSDD-v1.0 studied Faster R-CNN on the complete dataset and obtained an F1-score of 0.76 and a mAP of 0.748. Here, we have tested Mask R-CNN, which is an improvement on Faster R-CNN. As the training did not work on the complete dataset, we can not directly compare the performances. However, we notice that our results on the other datasets are poorer.

4.4.3.2 Visual difficulties

The difficulties encountered by Mask R-CNN are the same as those of YOLOv4, see Section 4.3.2.2, except that

- False positives identifying a ship on land are very rare. There seem to be fewer than with YOLOv4.

- One frequent problem is that several bounding boxes seem to have been produced for the same ship. Indeed, when there are a green and a red bounding box for the same ship or two red bounding boxes for it, it indicates that two predictions have been produced for this ship, see Figure 4.24.[6] This problem could probably be solved by adding to the implementation a system for filtering detections that overlap too much and keeping the one with the highest confidence score.

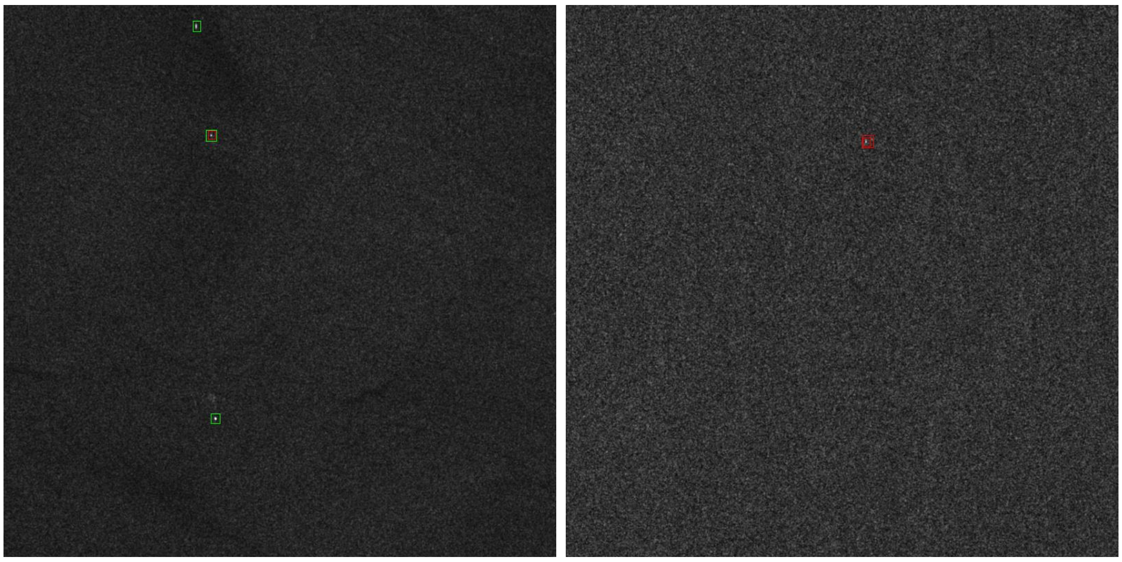


Figure 4.24: Images 10_1_5.jpg (left) and 12_2_21.jpg (right).

4.4.4 Threshold optimisation: reducing some difficulties

As with YOLO, the IoU and confidence thresholds can be modified to improve performance measures and reduce some visual difficulties. By default, they are set to 0.5 and 0.3, respectively. The same behaviour as for YOLO has been observed for the IoU threshold, see Section 4.2.4.1, so it has been set at 0.3. When studying the influence of the confidence threshold, it was observed that the performance measures (mAP and F1-score) increase when the threshold is greater than 0.3. In fact, with such a higher threshold, the number of false positives decreases significantly and there is no longer a problem of double detection for the same ship. The confidence threshold was therefore set at 0.7.

In this section, we present the effects on the (external) performance measures and on the visual difficulties. As with YOLO, the analysis focuses on PN dataset.

4.4.4.1 Performance measures

The performance measures of Mask R-CNN's final model on PN dataset when using the new and default thresholds are given in the table below. Modifying the thresholds significantly increases all performance measures, but mainly the F1-score and the precision. The resulting values obtained on the test set are quite good.

[6]As with YOLO, the green bounding boxes indicate true positives, the red ones false positives and the blue ones false negatives.

Thresholds		**Validation set**				**Test set**			
IoU	*Confidence*	*F1-score*	*precision*	*recall*	*mAP*	*F1-score*	*precision*	*recall*	*mAP*
0.3	**0.7**	**0.872**	**0.914**	**0.833**	**0.823**	**0.837**	**0.853**	**0.822**	**0.797**
0.5	0.3	0.718	0.629	0.836	0.799	0.697	0.656	0.731	0.714

4.4.4.2 Visual difficulties

Modifying the thresholds has almost eliminated double detection as well as the problem of false positives and false negative due to a lack of overlap. The remaining difficulties are listed below.

- A lot of ships are not detected near coasts or in harbours.
- Some tiny ships are difficult to detect.
- Small bright dots in the sea are identified as ships when they possibly are small pieces of land.
- Ships are not correctly identified when their contrast with the background is low.
- Some false negatives and false positives appear on the edge of some images. The reason for these poor detections is the same as explained previously.
- The parts of the land misidentified as ships are very rare.

Chapter 5

Comparison of optimised models

In this chapter, all the optimised models are compared. To this end, the performance measures obtained on the PN dataset (after threshold adjustment) are compared as well as the remaining visual difficulties. A brief comparison with the results obtained in the article on LS-SSDD-v1.0 dataset [61] follows. In addition, each model is applied to an image external to the LS-SSDD-v1.0 dataset to illustrate the generalisation ability of each model. Finally, the methods are compared based on their training curves and implementation.

5.1 Performance measures

The table below shows the performance measures of the optimised models of the three methods on the PN dataset.

Method	**Validation set**				**Test set**			
	F1-score	*precision*	*recall*	*mAP*	*F1-score*	*precision*	*recall*	*mAP*
YOLOv3	0.747	0.801	0.702	0.663	0.686	0.738	0.644	0.587
YOLOv4	0.784	0.828	0.746	0.716	0.819	**0.861**	0.782	0.755
Mask R-CNN	**0.872**	**0.914**	**0.833**	**0.823**	**0.837**	0.853	**0.822**	**0.797**

To assess these models, the test set is used. The performance measures of YOLOv3 are much poorer than those of YOLOv4 and Mask R-CNN. This was expected since one drawback of YOLOv3 is its difficulties to detect small objects. Consequently, the fourth version of YOLO is preferable to the third, which was expected too.

The other two methods have much more comparable performance measures. Apart from precision, Mask R-CNN performs better than YOLOv4. In other words, the number of false positives is slightly lower for YOLOv4, but the number of false negatives is slightly higher.

However, we can not directly conclude that Mask R-CNN is better. Indeed, the same 80/10/10 splitting proportion for the training/validation/test datasets was used, but the resulting subsets do not contain the same images. For example, the training set of Mask R-CNN does not include the same images as the training set of YOLOv4; the same applies to the validation and test sets. It is therefore worth investigating whether the performance measures vary significantly when the images are distributed differently in these subsets.

When another splitting is used for YOLOv4, the results on the test set are very similar than with the previous splitting. The mAP is slightly better, whereas the F1-score is a bit lower. This suggests that the model is relatively stable, since the performance measures vary only slightly. This is illustrated in the table below.

YOLOv4	**Validation set**				**Test set**			
	F1-score	*precision*	*recall*	*mAP*	*F1-score*	*precision*	*recall*	*mAP*
First splitting	0.784	0.828	0.746	0.716	**0.819**	**0.861**	0.782	0.755
Second splitting	**0.817**	**0.845**	**0.790**	**0.767**	0.801	0.811	**0.791**	**0.761**

By using another splitting for Mask R-CNN, the performance measures on the test set (except precision) decrease sharply, which seems to indicate that there is a large error rate. The performance measures quoted previously therefore appear to be overestimated compared with the actual performance measures of Mask R-CNN. This is illustrated in the table below.

Mask R-CNN	**Validation set**				**Test set**			
	F1-score	*precision*	*recall*	*mAP*	*F1-score*	*precision*	*recall*	*mAP*
First splitting	**0.872**	**0.914**	**0.833**	**0.823**	**0.837**	**0.853**	**0.822**	**0.797**
Second splitting	0.855	0.887	0.826	0.813	0.760	0.847	0.689	0.675

Given that Mask R-CNN has a large margin of error, which causes it to overestimate itself, whereas YOLOv4 seems relatively stable, the latter seems preferable in terms of performance measures. Nevertheless, it can not be said that one is clearly superior to the other.

5.2 Comparison with results of the LS-SSDD-v1.0 paper

Let us briefly compare our results with those achieved by the creators of LS-SSDD-v1.0, who worked exclusively on the complete dataset. [61]

Before optimising the thresholds, we obtained better results for YOLOv3 on the complete dataset, see Section 4.2.3.2. Now that the thresholds have been optimised, the difference is even greater: we obtained an F1-score of 0.686 and a mAP of 0.587, compared to an F1-score of 0.40 and a mAP of 0.246 in the article. It should be noted that we obtained our final model on PN and not on the complete dataset like them.

We assume that our results are much better than theirs because they have not optimised their hyperprameters. Furthermore, in Section 4.2.3.2, we did not observe a large difference in performance measures depending on the training dataset (in particular PN and complete), so the difference in performance measures probably does not come from the set on which it was trained.

Before optimising the thresholds, we observed that our results for Mask R-CNN were poorer than their results for Faster R-CNN, see Section 4.4.3.1. This comparison could not be made on the complete dataset because the Mask R-CNN training was not possible on it. However, regardless of the dataset (P, PN or P2N), the results were worse. Now that the thresholds have been optimised (on PN dataset), we have an F1-score of 0.837 and a mAP of 0.797, compared to an F1-score of 0.76 and a mAP of 0.748 in the article (on the complete dataset). So, our results for Mask R-CNN are better than those of the article for Faster R-CNN. However, if we compare the results obtained with the second splitting of Mask R-CNN, the conclusions are different: we obtain the same F1-score value, but a poorer mAP.

Theoretically, Mask R-CNN performs better than Faster R-CNN as it is an improvement of this method. Because of the instability of Mask R-CNN, we can not conclude that it outperforms Faster R-CNN. Nevertheless, the two methods are comparable in terms of performance measures.

Finally, we observe that our results for YOLOv4 (on PN dataset and regardless of the splitting) are better than their results for Faster R-CNN (on the complete dataset).

5.3 Visual difficulties

Based on the conclusions reached in Sections 4.3.3.2 and 4.4.4.2, the same visual difficulties are encountered by both methods and it is not possible to determine whether one handles some situations better than the other, except that Mask R-CNN seems to make fewer predictions on land, but these false positive detections can be removed using a land-sea mask for both methods. Consequently, this difference is not sufficient to choose between both methods.

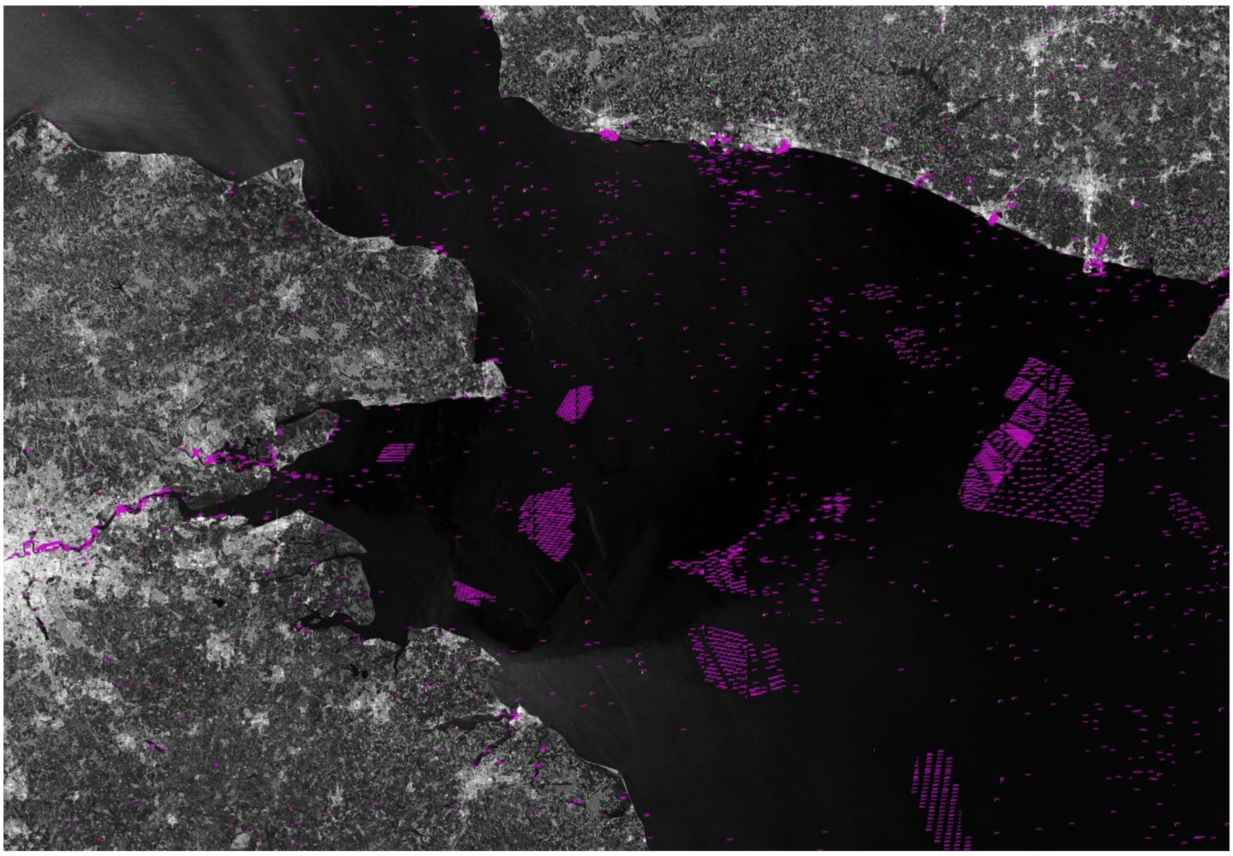
We have just discussed the visual difficulties identified on the test sets of the LS-SSDD-v1.0 dataset. Now that the models have been optimised, a new SAR image external to this dataset can be tested. For this purpose, a large-scale SAR image of the Belgian coast taken by Sentinel-1 on the 4th of August 2022 was selected.[1] However, there are no ground truth annotations available for ship positions. The analysis is therefore purely visual and rather subjective.

The three methods have been tested on this image, see Figures 5.1 and 5.2, where the predicted bounding boxes are shown in purple. These boxes are very small because it is a large-scale image and so the ships are proportionally very small compared with the size of the image. As shown in Figure 5.2, the display of predictions for Mask R-CNN is much less visible due to its implementation.

YOLOv3 makes a lot of false predictions inland and contrary to the observations made on the test set, it makes a lot of predictions on the coasts and in harbours. There are also large clusters of ships detected in the open sea. In fact, these correspond to wind farms. According to [61], the LS-SSDD-v1.0 dataset does not contain wind farms. If the dataset on which the model has been train would have contain images with wind farms, the model might have learned to distinguish them from ships.

These wind farms are also detected by YOLOv4 and Mask R-CNN. To remove these predictions, it would be possible to filter them using the positions of wind farms, which are easily available. These two methods provide similar predictions on this external image: they give almost no detection on land and the behaviour of the predictions on open sea, on the coasts and in harbours are very similar.

[1]This image was selected by the first student working on the project and was obtained with the Alaska Satellite Facility. [2]



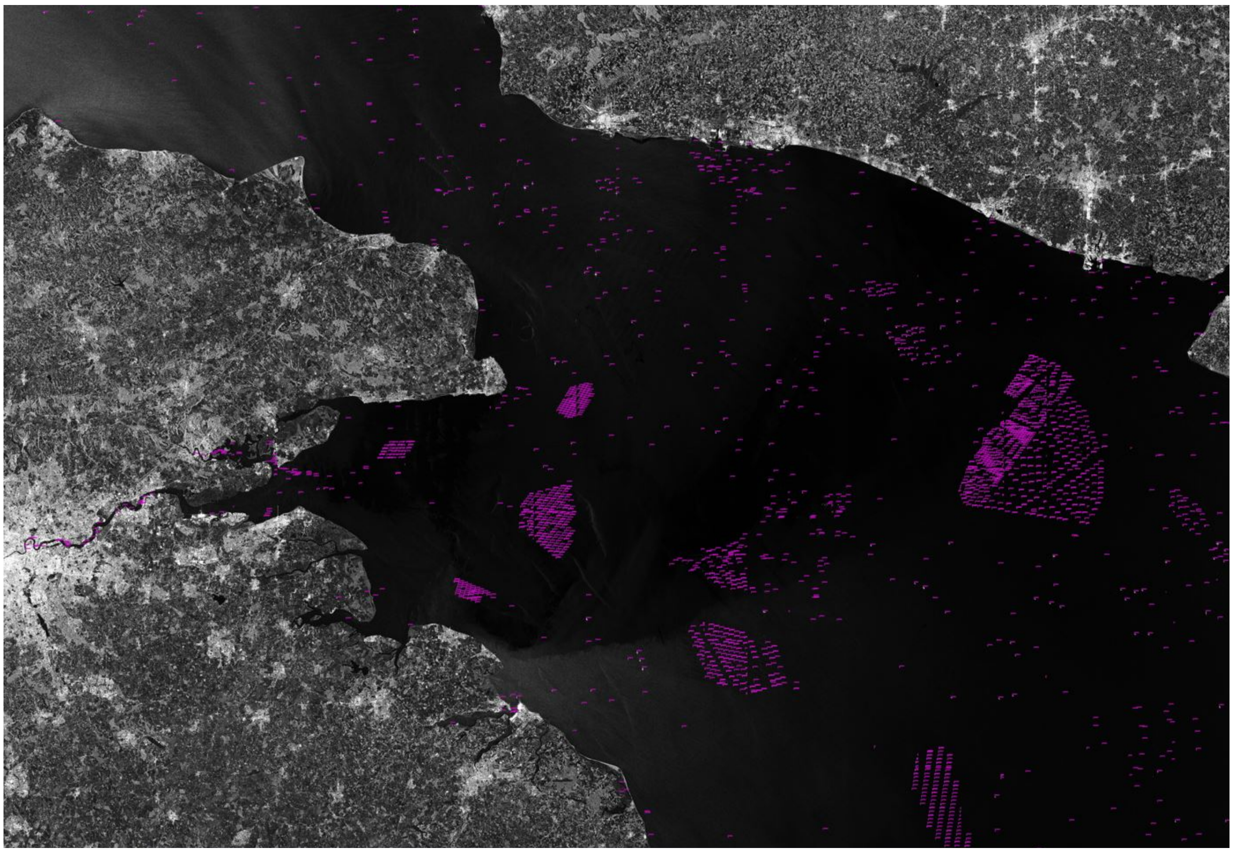


Figure 5.1: Test of YOLO methods (YOLOv3 top and YOLOv4 bottom) on a large-scale SAR image of the Belgian coast taken on the 4th of August 2022: predicted bounding boxes are displayed in purple.

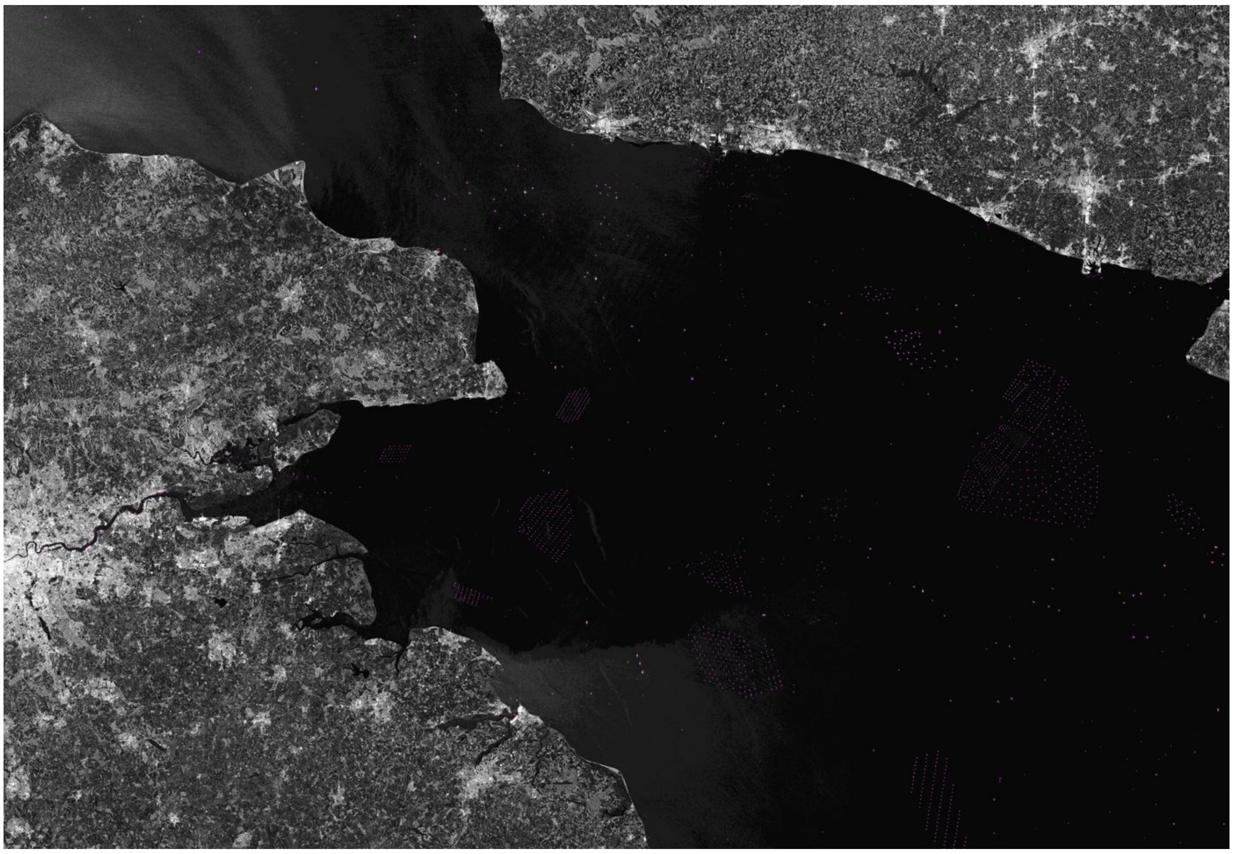


Figure 5.2: Test of Mask R-CNN on a large-scale SAR image of the Belgian coast taken on the 4th of August 2022: predicted bounding boxes are displayed in purple.

5.4 Training curves and implementation

The graphs of the loss and mAP curves obtained during training can also be compared. The loss curve of Mask R-CNN has a decreasing behaviour but it does not decrease much compared to the one of YOLO. Moreover, its mAP curve stabilises quickly and therefore no longer tends to increase, unlike the one of YOLO. In fact, despite the large number of iterations, its mAP curve is not totally stable and still seems likely to increase.

More details of the implementation will be given in the next chapter, but Mask R-CNN is implemented in Python while YOLO uses the framework Daknet which is implemented in C. This implies that Mask R-CNN has performance and memory disadvantages over YOLO. Indeed, Python is slower than some other languages because it is not a compiled language but an interpreted language. In addition, the maximum possible batch size for Mask R-CNN is 12, whereas the subdivisions hyperparameter in YOLO allows the use of a much larger batch size, such as 64 or 128. From theory, a larger batch size should result in better performance, which means that YOLO has an advantage over Mask R-CNN. Finally, there seem to be more hyperparameters to optimise in YOLOv4 than in Mask R-CNN, which could further improve the model.

In conclusion, YOLOv3 performs poorly, while YOLOv4 and Mask R-CNN are comparable. As the error rate was higher for Mask R-CNN, this would suggest greater stability for YOLOv4 and therefore better results. In addition, in terms of implementation, training and hyperparameters, it might be possible to further optimise YOLOv4. However, it is not clear that one method is preferable to another.

Chapter 6

Implementation

This chapter provides a brief overview of the various codes used to train the different methods. Concerning YOLO, no implementation has been made since Darknet provides everything necessary for training the model and evaluating it, hence the use of Darknet's performance measures.

Regarding Mask R-CNN, as the name suggests and as explained in the theoretical part, a mask is required for each image of the dataset. However, LS-SSDD-v1.0 dataset does not contain any mask. It was therefore necessary to create them. The implementation of Mask R-CNN was adapted for the LS-SSDD-v1.0 dataset from a PyTorch[1] tutorial. [43] The contributions made are briefly presented in this chapter. Finally, the implementation of the external performance measures was adapted from the code provided by the first student who worked on this project so that it could be used for YOLO and Mask R-CNN. No further details about this implementation are provided.

6.1 Mask creation

In this section, we describe the mask creation process, provide some illustrations of the masks created and explain the difficulties encountered during the process, as well as the conversion to the Mask R-CNN format.

6.1.1 Creation process

Recall that the LS-SSDD-v1.0 dataset is composed of images and their associated annotation file which contains the coordinates of corresponding bounding boxes.

There are positive and negative images, i.e. images that contains at least one detected object or none at all. For each positive image, the positions of the detected ships are available (in the annotation file). Hence, one bounding box is associated to each ground truth detection. The mask creation process is handled separately on each bounding box.

First, the complete image is converted to greyscale. In our case, satellite images are already in a greyscale format. The image is then represented by an array of the same dimension as the image with values between 0 and 255, where 0 corresponds to black, 255 to white and the intermediate values correspond to shades of grey. The image in Figure 6.1 will serve as an illustration.

[1]PyTorch is an open source Python framework for machine learning, based on the Torch library and developed by Meta.

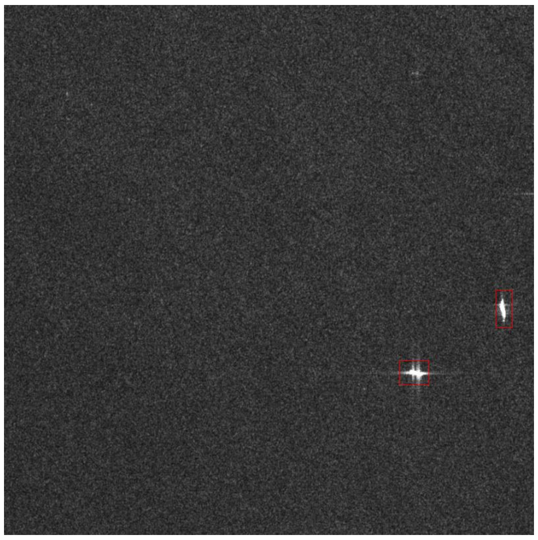


Figure 6.1: Image 03_5_11.jpg of the LS-SSDD-v1.0 dataset: the bounding boxes of two detected ships are displayed in red.

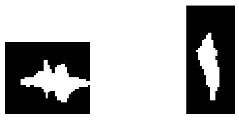
Then, for a selected image, the following steps are performed separately on each sub-image associated to a bounding box to create the corresponding mask.

- Work on the part of the image restricted to the bounding box. In this example, the steps will therefore be carried out independently on the two sub-images below.

- Adjust the range of values in this sub-image to the interval $[0, 255]$. The purpose of this step is to increase the difference in light intensity between the ship and its surroundings to make it brighter. This is useful in the case of detections that are too small or when the contrast between the ship and the background is low. Without this step, the mask created could have been smaller than the detection or would have completely disappeared.
- Dilate the sub-image to thicken or increase the size of the object. For example, if an object is separated into close parts, this permits to group them. This step has two parameters: the kernel size and the number of dilation steps. The kernel, a matrix of ones, is slid over the given sub-image. If the part of the image it covers contains a 255, all the pixels covered are assigned the value 255. This step increases the size of the white regions and therefore the size of the objects. The larger the kernel size, the thicker the object will be. This step can be performed several times by specifying it in the parameter of the number of dilation steps, whose default value is 1. The more dilation steps are performed, the more details of the object's contours will be lost. However, the masks must remain precise enough. This step is nevertheless important here to avoid the mask sticking too much to small imperfections of the ship's contour. Here, the chosen kernel size is $(2, 2)$ and only one iteration of this process is done.

- Blur the image to remove the noise. This step has one parameter: the size of the kernel, which is a matrix of ones that similarly slides over the sub-image. As the kernel scans the image, the values of the pixels covered by the kernel are averaged. This average value is then assigned to each pixel covered by the kernel. The larger the kernel size, the more blurred the object and therefore the less noise. In other words, this step smoothes the contours of the object. Here, since the contours of ships are generally quite noisy due to their light intensity, a kernel of size $(2, 2)$ has been used.

- Finally, apply a binary threshold to provide a binary image. Each value smaller than the selected threshold is set to 0 while the other ones are set to 1. This provides a binary array leading to a black and white image, which corresponds to the final mask. A default value for the threshold is 127. The closer the threshold is to 255, the smaller the white regions; and the closer the threshold is to 0, the larger they are. Since the default value leads to too large masks or masks including too much noise around the ships, the threshold was therefore set to 160.



Once these steps have been applied separately on each detection of the selected image, they are inserted in the right positions in a black image. By doing this on the example, the following mask is produced.

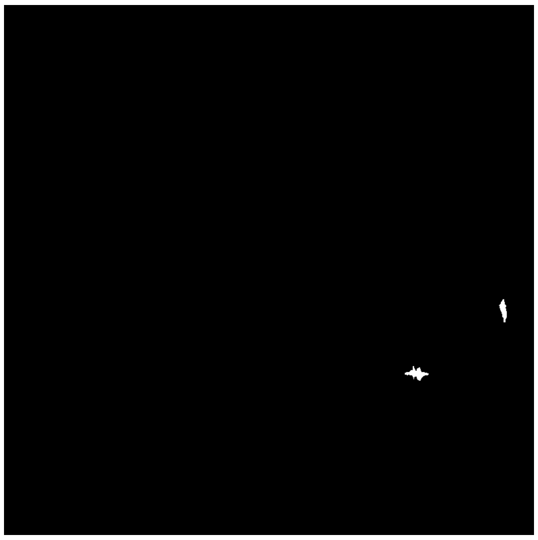


Figure 6.2: Mask created for image 03_5_11.jpg.

The mask creation process described above should be followed for positive images, i.e. those containing at least one detected ship. For negative images containing no detection, empty masks are required. For this purpose, the mask creation process provides images that are completely black.

6.1.2 Illustration

Figures 6.3 and 6.4 provide some examples of images, on which the bounding boxes are displayed in red, as well as their associated mask created using the process described above.

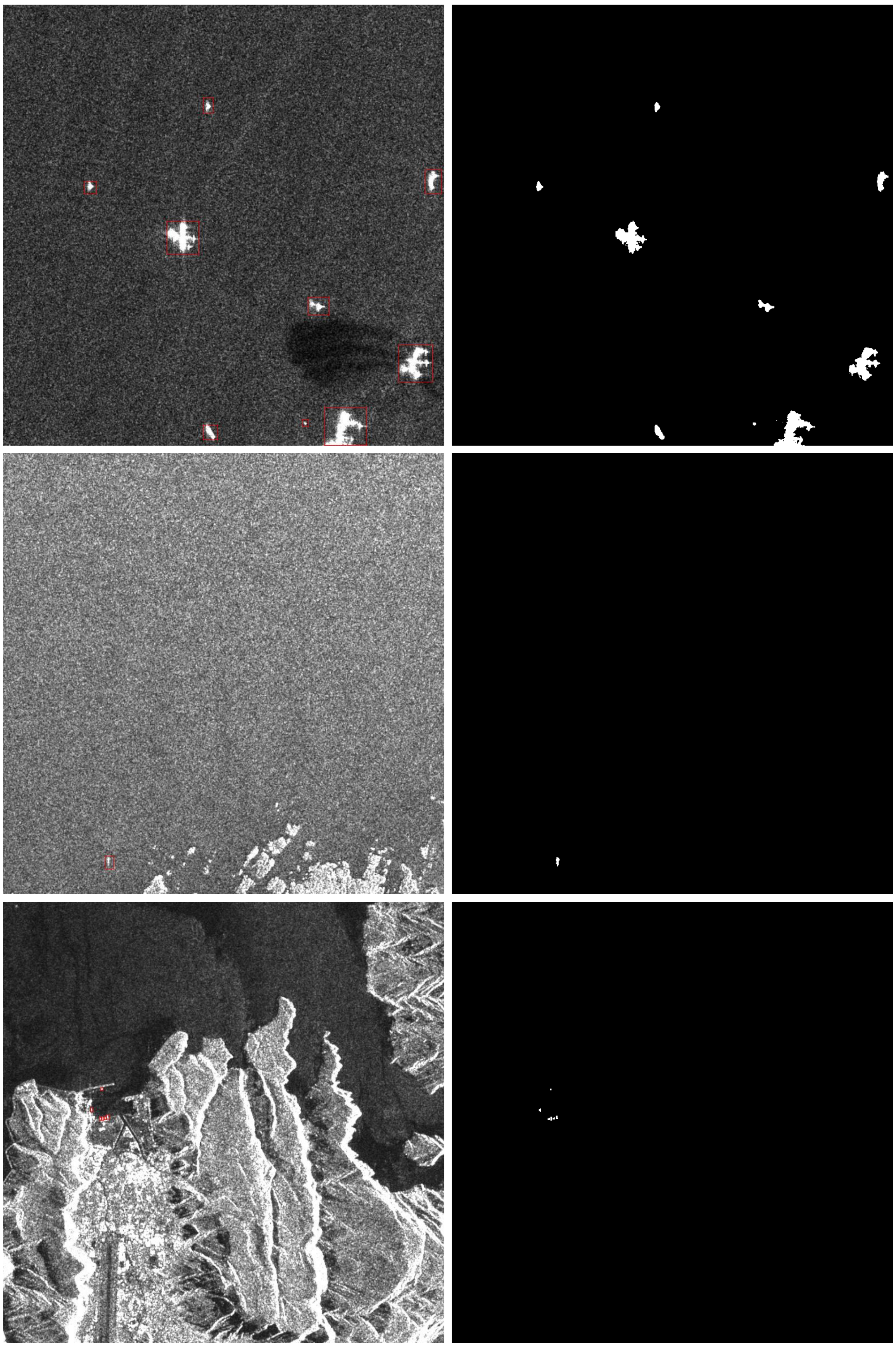


Figure 6.3: Images 05_9_18.jpg, 03_13_28.jpg, 04_12_15.jpg and their masks.

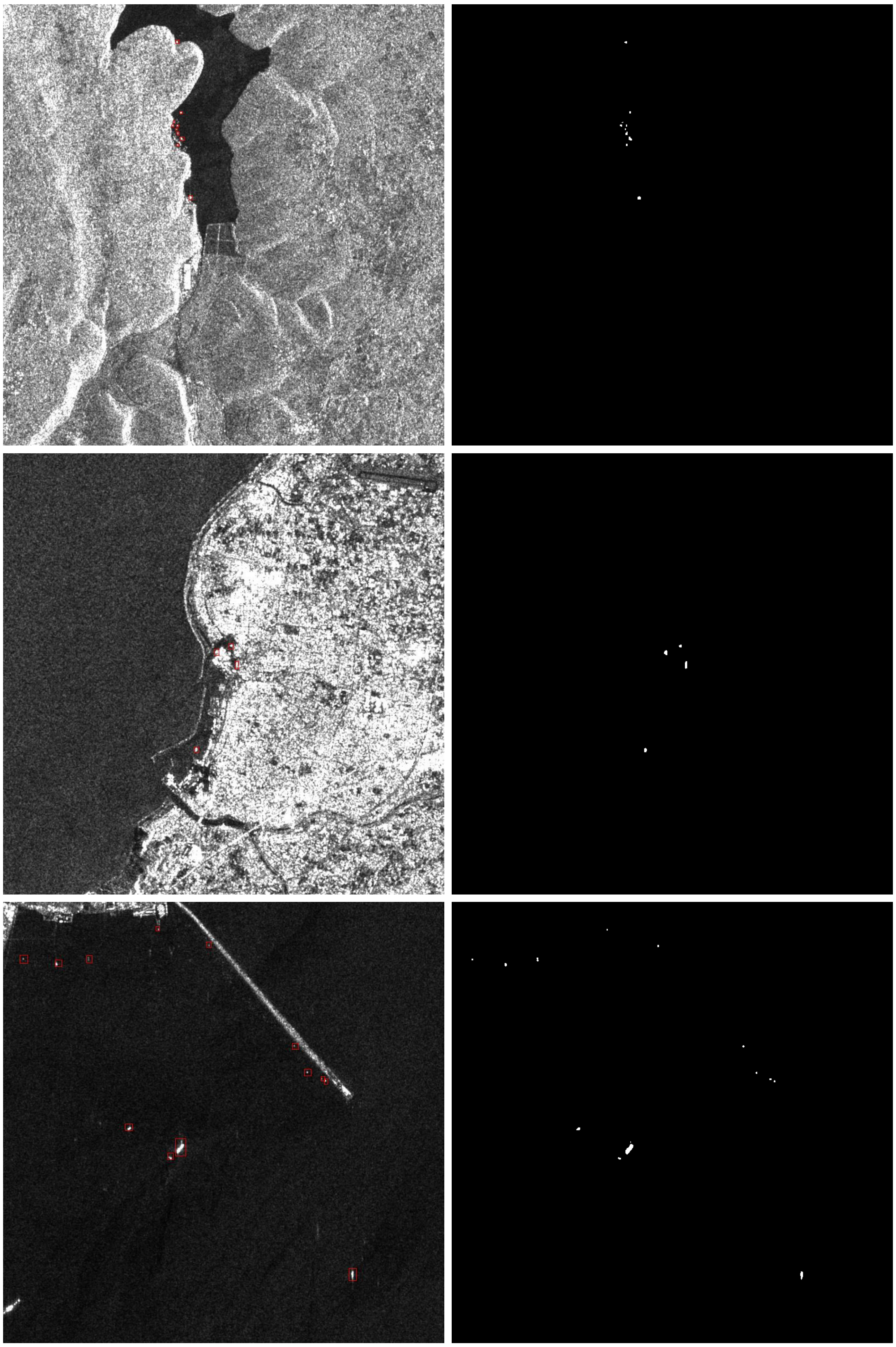


Figure 6.4: Images 02_16_13.jpg, 01_14_28.jpg, 01_19_10.jpg and their masks.

6.1.3 Difficulties

The table below summarises the selected parameter values of each step of the mask creation process described in Section 6.1.1.

Step	**Parameter**	**Selected value**
Dilatation	kernel size	$(2, 2)$
	number of dilatation steps	1
Blurring	kernel size	$(2, 2)$
Binary threshold	threshold	160

These values lead to masks of good quality in a "normal" situation. For instance, the masks associated to the images presented in the previous section are of good quality. Indeed, they cover the ships in a similar way to what is distinguishable to the human eye. They maintain the same shape and approximately the same size as the ship. However, creating good masks is more complicated in several specific situations. The values selected above provide good compromises in these difficult situations, but some of the masks created are still of poor quality. This section presents situations that lead to difficulties when creating masks.

- When the contrast between the ship and the background is low, it is more difficult to create a good mask. With the chosen parameters, this situation is usually handled well, as shown in image 03_13_28.jpg in Figure 6.3, but can lead to poor quality masks as illustrated below in Figure 6.5. Indeed, in the bottom left of this image, the contrast between the ships and the background is quite low, so the masks created include a lot of noise around the ship. These masks are particularly poor as they correspond approximately to the bounding box that frames the ship.

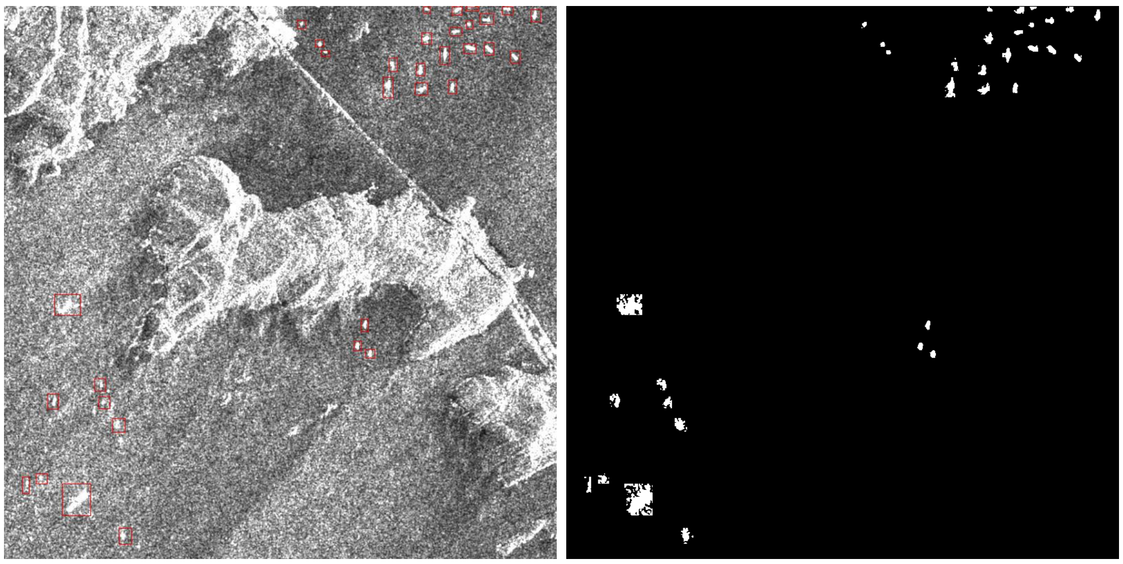


Figure 6.5: Image 11_13_29.jpg and its mask.

- When several bounding boxes are close to or against each other, it may be difficult to distinguish the ships. In practice, most of the time, when they are distinguishable to the human eye, the masks created are clearly distinct, as for image 02_16_13.jpg in Figure 6.4. When they are not distinguishable, sometimes the masks are distinct though, like image 04_12_15.jpg in Figure 6.3, but in general, the masks are joined together as illustrated below in Figure 6.6.

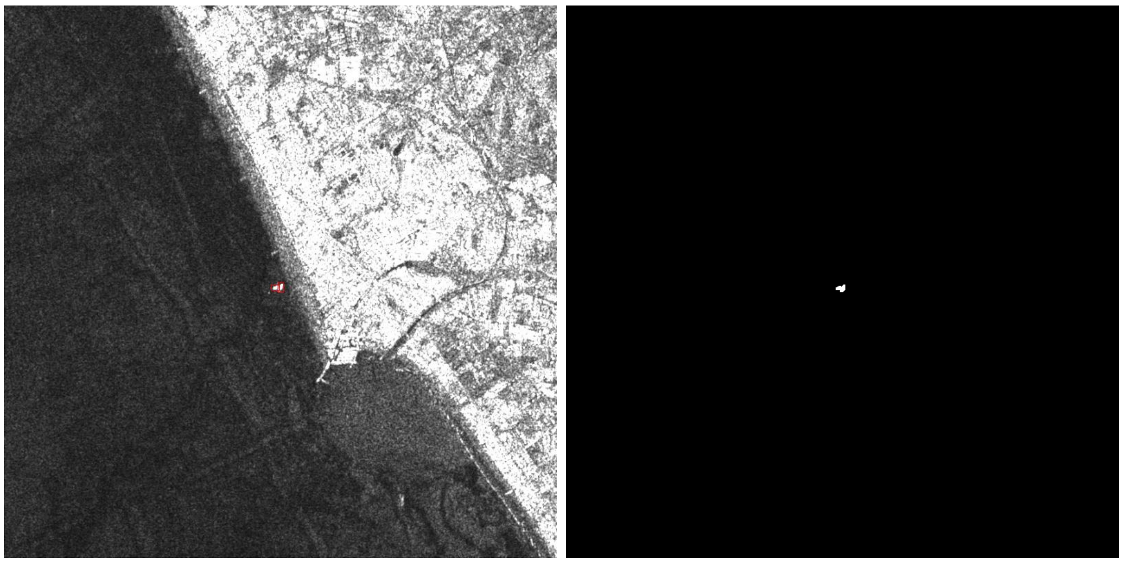


Figure 6.6: Image 02_6_28.jpg and its mask. Two detections are against each other and are not distinguishable to the human eye. The corresponding masks are joined.

- When ships are close to the coast, they may be confused with the brightness of the land. In some situations, the ships are still distinguishable and the masks are good, like image 01_14_28.jpg in Figure 6.4. Otherwise, as below in Figure 6.7, the mask created contains the ship as well as the background brightness. In this case, these masks are poor as it is not possible to distinguish the ship from the land, see the bottom right of the image.

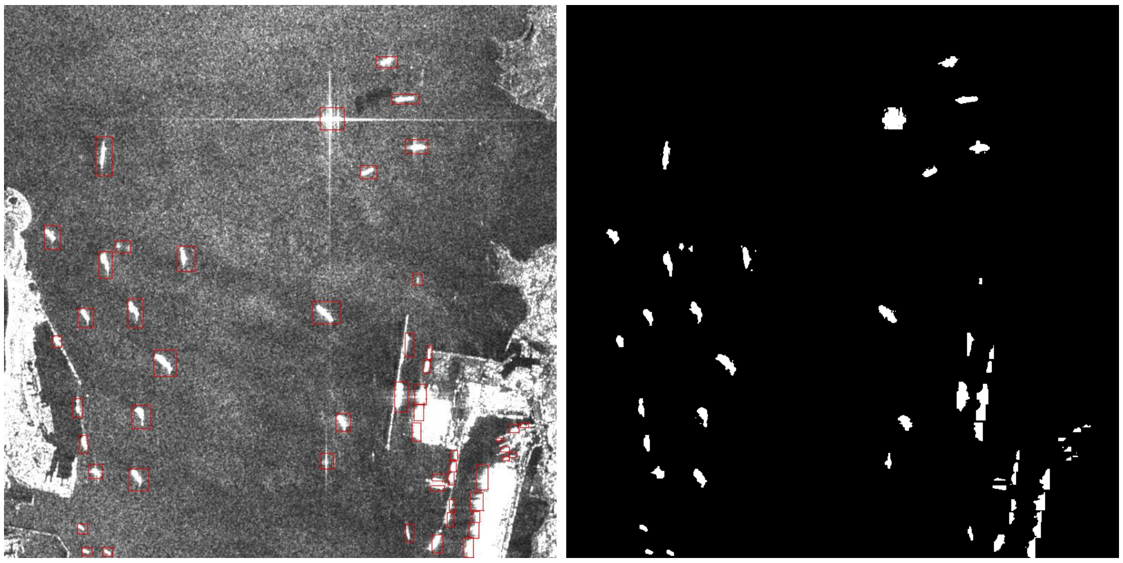


Figure 6.7: Image 15_15_22.jpg and its mask.

- If a ship is split into two distinct parts that are far apart, they may also be separated on the mask. The dilation applied during the mask creation process allows two parts that are relatively close to be joined. The case below, see Figure 6.8, is an example of a detection split into two distant parts whose mask is also disjoint.

As mentioned earlier, the parameters chosen in the mask creation process enable a good trade-off between these difficulties. Indeed, increasing the threshold parameter results in better masks when the contrast is low, but small detections tend to have a small mask or even a disappearing mask. Increasing the size of the dilation kernel would split fewer ships

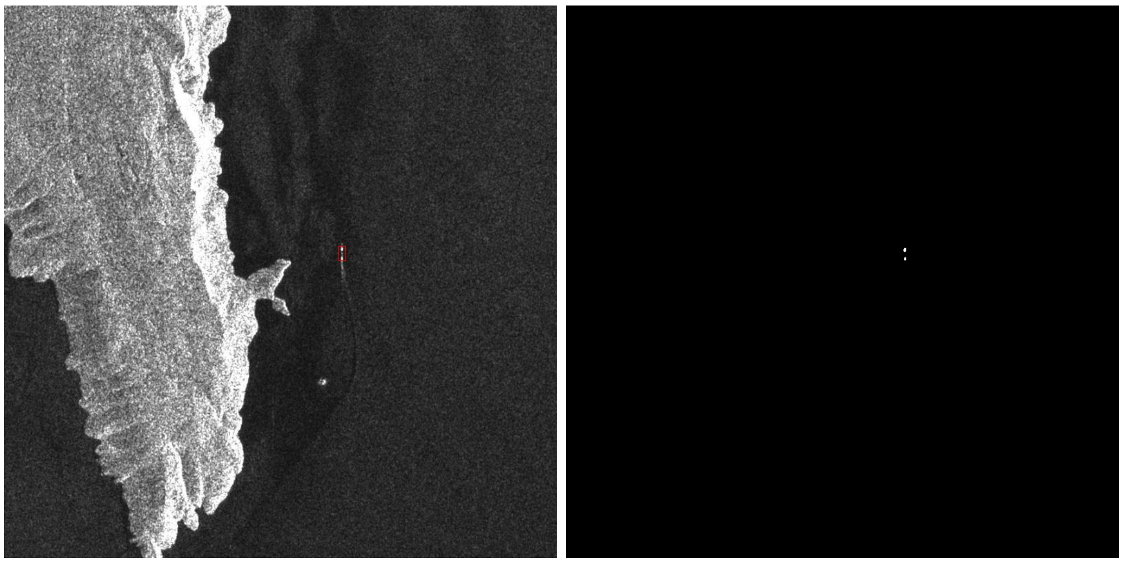
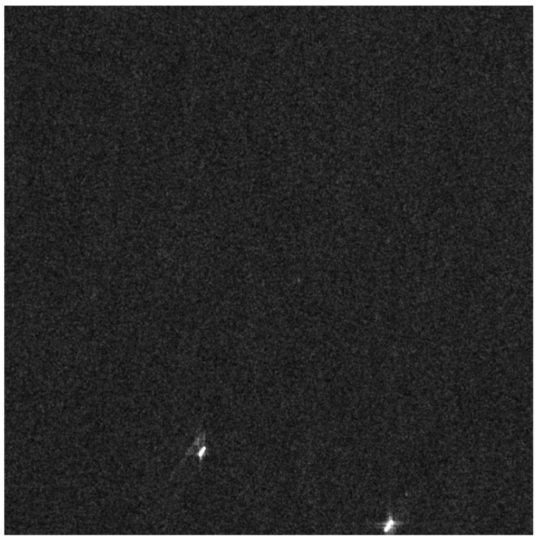


Figure 6.8: Image 02_14_11.jpg and its mask.

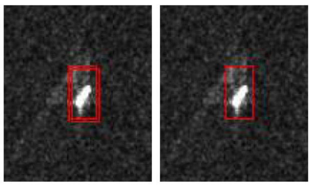
in two, but all masks would tend to be too thick given the size of the ship and close detections would be more frequently joined. Finally, increasing the kernel size of the blur would cause too much detail of the ship contours to be lost, especially the smaller ones.

In conclusion, the parameter values have been selected for good behaviour in normal situations and for a good compromise in difficult situations.

Remark 6.1.1. During the mask creation process, we found a problem related to the bounding box annotations of an image in the dataset. For that image, one mask is completely covered by another. In fact, we found two bounding boxes framing the same ship revealing an annotation problem. The bounding box associated to the fully covered mask has therefore been completely removed from the list of bounding boxes for the Mask R-CNN training. Figure 6.9 (a) illustrates the problematic image. The detection on the left has two associated boxes, one of them is deleted, see Figure 6.9 (b).



(a)



(b)

Figure 6.9: (a) Image 12_16_16.jpg. (b) On the left: problematic bounding boxes in image 12_16_16.jpg: two bounding boxes are almost superimposed. On the right: suppression of one bounding box to retain only one.

6.1.4 Mask R-CNN format

The mask creation process described in Section 6.1.1 provides binary masks. This means that for a given image, all of its detections are associated to value 1 of the mask. However, the masks that feed the Mask R-CNN algorithm require a particular format. The values of the mask associated to different detections must be distinguishable. For this purpose, a different value is associated to each detection. This is illustrated in Figure 6.10 with the mask of image 05_9_18.jpg, see Figure 6.3. The values associated to the different detections correspond to different shades of grey.

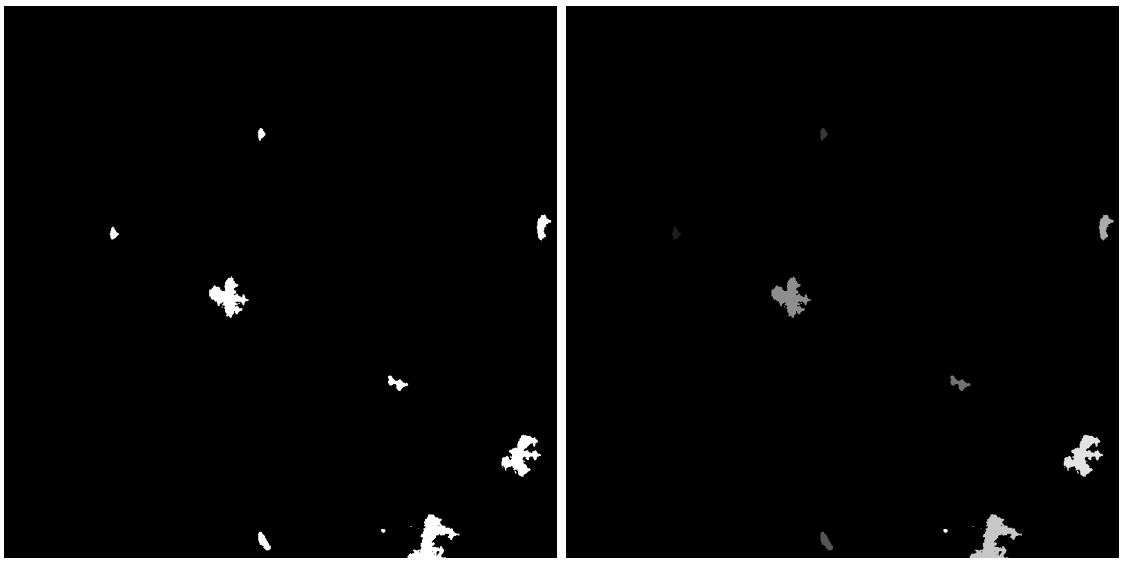


Figure 6.10: Black and white (left) and coloured (right) masks of image 05_9_18.jpg.

6.2 Mask R-CNN implementation

The implementation of the training and testing processes of Mask R-CNN has been adapted from a PyTorch tutorial for pedestrian detection and segmentation. [43] Each part of the code relies on a Python class that has been modified for the LS-SSDD-v1.0 dataset. The contributions made to the implementation are the following:

- Thanks to the mask creation process, the code has been adapted to any dataset with only bounding boxes annotations in Pascal VOC format (*xml* format).
- The values of the loss and mAPs (bounding box and segmentation mAPs as well as their average value) during training have been extracted from a training log file in order to create graphs representing their evolution during the learning process.
- In the initial code, the training stops when the predetermined number of epochs has been reached and the model retained was the one obtained in the last iteration, but this has been modified in order to retain the model that led to the best mAP during training.
- A step applying the non-maximum suppression algorithm has been added to the test code: when several boxes have an IoU greater than a given threshold, only the box with the highest confidence score is retained.

Chapter 7

Problems

In this chapter, we briefly explain the problems encountered during the practical part of the project since several factors slowed down the progress of this part.

First, a new graphics card was expected, so it was preferable to wait for it to be installed before starting the practical part. Then, there were problems with CUDA when optimising YOLO: a parameter (cuDNN) that should speed up the training was not working properly. Thus, training times were doubled or even tripled. It took some time before solving this problem. Finally, adapting the tutorial for Mask R-CNN was not straightforward and also took quite a while.

Chapter 8

Conclusion

The aim of this master's thesis was to determine a deep learning program adapted to ship detection from SAR images. For this purpose, the LS-SSDD-v1.0 dataset selected by the first student working on this project was used and several subsets were created, including one containing only positive detections (P dataset) and two containing both positive and negative detections (PN and P2N). Three methods were considered: versions 3 and 4 of YOLO and Mask R-CNN. Their hyperparameters were optimised manually, mainly on P dataset. The optimised models were then evaluated and compared, along with their visual difficulties. Finally, they were applied to a new large-scale SAR image of the Belgian coast, allowing their predictions to be compared on an image external to the dataset.

It was concluded that YOLOv3 performs poorer than YOLOv4, as theoretically expected. The aim here was to confirm this fact. It was also shown that by tuning the hyperparameters of YOLOv3, much better performance than those obtained by the authors of LS-SSDD-v1.0 dataset was achievable. The comparison between YOLOv4 and Mask R-CNN is more subtle. Mask R-CNN has a large margin of error depending on the splitting of the dataset used, whereas YOLOv4 seems more stable. Both models encounter the same visual difficulties on the test set and on the external image, which makes it difficult to prefer one method over the other. The predictions on the external image were compared only visually. A more formal approach to verify the quality of the predictions would be to use the AIS messages and compare the positions of the ships detected by the model with the positions indicated by the AIS messages. Finally, there seems to be more scope for further tuning the hyperparameters of YOLOv4 than those of Mask R-CNN. In other words, improving YOLOv4 model seems possible while improving Mask R-CNN seems more limited.

In conclusion, a deep learning algorithm has been successfully determined for ship detection from SAR imagery. Indeed, YOLOv4 and Mask R-CNN are both good candidates. Nevertheless, if a single method should be selected, it would be YOLOv4 due to its greater stability and its better possibility of improvement.

Chapter 9

Limitations

In this chapter, we describe the limitations of our work.

Firstly, the same 80/10/10 splitting ratio was used, but the images included in each subset were not the same. This means that YOLOv4 and Mask R-CNN are neither trained nor tested on the same images. When the model optimisation was complete, another splitting into subsets was used and it was found that the performance measures of YOLOv4 were relatively stable, unlike those of Mask R-CNN. As a result, the dataset was not sufficiently large for such variations in performance to be avoided. In order to compare the three methods fairly, the same train, validation and test sets should have been used. An even fairer approach would have been to train the different models using K-fold cross-validation, but this was tricky to implement and would have considerably increased the training time. Finally, to accurately compare their performance measures, a nested cross-validation could have been applied, but again, this would have been much more time-consuming.

The study of the local hyperparameters of each method as well as the global hyperparameters of Mask R-CNN were studied on the P dataset. Indeed, the global hyperparameters of YOLO had initially been studied on P and PN datasets, but as it was time-consuming to do it on both and the performance measures on P were better, the latter was selected. However, according to the theory, it is better to train on a balanced dataset because it is generally better to generalise afterwards. It would therefore have been better to work on PN dataset.

Finally, there are methods for automating the hyperparameter optimisation process, such as grid and random search. Such methods require a range of values to test for each hyperparameter and return the most appropriate value. However, for a first contact with the field of machine learning, the idea was rather to do this manually in order to better understand the process behind it and the influence of each hyperparameter.

Part III

Appendix

A Taylor approximation for gradient descent

In Section 2.2.2, the loss function was approximated around $\boldsymbol{\theta}_0$ using the first-order approximation of Taylor's theorem as

$$\hat{L}(\boldsymbol{\theta}_0 + \boldsymbol{\epsilon}) = L(\boldsymbol{\theta}_0) + \epsilon^T \nabla_\theta L(\boldsymbol{\theta}_0) + \frac{1}{2\gamma}||\boldsymbol{\epsilon}||^2 \quad \text{with } \boldsymbol{\epsilon} \to 0,$$

where $\boldsymbol{\theta}_0$ denotes the initial value of the model parameter and γ the learning rate. This section provides an explanation of the derivation of this expression. For this purpose, recall the statement of Taylor's theorem.

Theorem A.1 (Taylor's theorem - first-order approximation). *Suppose $X \subseteq \mathbb{R}^n$ is open, $\boldsymbol{x} \in X$ and $f : X \to \mathbb{R}$ is of class C^2. Then, for $\boldsymbol{h} \in \mathbb{R}^n$,*

$$f(\boldsymbol{x}+\boldsymbol{h}) = f(\boldsymbol{x}) + \boldsymbol{h}^T \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) + O(||\boldsymbol{h}||^2) \quad \textit{as } \boldsymbol{h} \to 0.$$

In our case, we are working with a loss function $L : \mathbb{R}^d \to \mathbb{R}$. Hence, to apply Taylor's theorem, the chosen loss function must be taken of class C^2. By applying it to this function around $\boldsymbol{\theta_0} \in \mathbb{R}^d$, we get

$$L(\boldsymbol{\theta}_0 + \boldsymbol{\epsilon}) = L(\boldsymbol{\theta}_0) + \boldsymbol{\epsilon}^T \nabla_\theta L(\boldsymbol{\theta}_0) + O(||\boldsymbol{\epsilon}||^2) \quad \text{as } \boldsymbol{\epsilon} \to 0.$$

Since $||\boldsymbol{\epsilon}||^2 \to 0$ and by taking a small scalar $\gamma > 0$, the following approximation is used instead

$$\hat{L}(\boldsymbol{\theta}_0 + \boldsymbol{\epsilon}) = L(\boldsymbol{\theta}_0) + \epsilon^T \nabla_\theta L(\boldsymbol{\theta}_0) + \frac{1}{2\gamma}||\boldsymbol{\epsilon}||^2 \quad \text{with } \boldsymbol{\epsilon} \to 0.$$

This expression is exactly the one used in Section 2.2.2.

B Convolutional neural networks

In Section 2.2.5, convolutional neural networks (CNN), composed of convolutional, pooling and fully connected layers, were presented. For this purpose, one-dimensional and 3-dimensional convolutions were defined, as well as maximum and average pooling. In the first section, some examples and figures are given to clarify the definitions of convolution. In the second one, there will also be for pooling.

B.1 Convolutional layer

Convolutional layers have already been presented in Section 2.2.5.1. In particular, one-dimensional convolution was defined as follows.

Definition B.1. For w, $W \in \mathbb{N}$ such that $w \leq W$, given an input vector $\mathbf{x} \in \mathbb{R}^W$ and a vector $\mathbf{u} \in \mathbb{R}^w$, called the *convolutional kernel*, the **discrete convolution $\mathbf{x} * \mathbf{u}$** is a vector of size $W - w + 1$ such that

$$(\mathbf{x} * \mathbf{u})[i] = \sum_{m=1}^{w} \mathbf{x}_{m+i-1}\, \mathbf{u}_m,$$

where $i \in \{1, ..., W - w + 1\}$.

The following example illustrates this definition.

Example B.1. If we are given

- $\mathbf{x} = (1, 4, -1, 0, 2, -2, 1, 3, 3, 1)^T \in \mathbb{R}^W$ with $W = 10$ and
- $\mathbf{u} = (1, 2, 0, -1)^T \in \mathbb{R}^w$ with $w = 4$,

the convolution $\mathbf{x} * \mathbf{u}$ is a vector of size $W - w + 1 = 10 - 4 + 1 = 7$. Let us compute the first component of this vector. It is given by

$$(\mathbf{x} * \mathbf{u})[1] = \sum_{m=1}^{4} \mathbf{x}_m \mathbf{u}_m = 1.1 + 4.2 + (-1).0 + 0.(-1) = 9.$$

In other words, this is equivalent to computing the element-wise product between the vectors $(1, 4, -1, 0)^T$ and $\mathbf{u} = (1, 2, 0, -1)^T$. The second component is computed as

$$(\mathbf{x} * \mathbf{u})[2] = \sum_{m=1}^{4} \mathbf{x}_{m+1} \mathbf{u}_m = 4.1 + (-1).2 + 0.0 + 2.(-1) = 0,$$

which is equivalent to computing the element-wise product between the vectors $(4, -1, 0, 2)^T$ and $\mathbf{u} = (1, 2, 0, -1)^T$. The third component is computed as

$$(\mathbf{x} * \mathbf{u})[3] = \sum_{m=1}^{4} \mathbf{x}_{m+2} \mathbf{u}_m = (-1).1 + 0.2 + 2.0 + (-2).(-1) = 1.$$

This is equivalent to computing the element-wise product between the vector $(-1, 0, 2, -2)^T$ and the kernel $\mathbf{u}$. By computing subsequent in the same way, we obtain

$$\mathbf{x} * \mathbf{u} = (9, 0, 1, 3, -5, -3, 6)^T.$$

Remark B.1. In fact, each element of a convolution corresponds to the element-wise product between a vector sliding on the input and the kernel, where the sliding vector has the same size as the kernel.

This definition was then generalised for 3-dimensional tensors, as follows.

Definition B.2. Considering

- *h, H, w, W, $C \in \mathbb{N}$ such that $h \leq H$ and $w \leq W$,*
- *a 3D-tensor $\mathbf{X} \in \mathbb{R}^{C \times H \times W}$, called the* input feature map*, and*
- *a filter $\mathbf{U} \in \mathbb{R}^{C \times h \times w}$, which slides across the input feature map, along its height and its width. The size of the filter determines the size of the receptive field, which is $h \times w$. The* receptive field *is defined as the region of the input feature map to which the layer has access and from which it extracts features.*

The **convolution $\mathbf{X} * \mathbf{U}$** outputs a 2D-tensor $\mathbf{O}$ of size $(H - h + 1) \times (W - w + 1)$, called the *output feature map*. The tensor $\mathbf{O}$ is defined as

$$\mathbf{O}_{j,i} = \sum_{c=1}^{C} (\mathbf{X}_c * \mathbf{U}_c)[j, i] = \sum_{c=1}^{C} \sum_{n=1}^{h} \sum_{m=1}^{w} \mathbf{X}_{c,\, n+j-1,\, m+i-1}\ \mathbf{U}_{c,\, n,\, m},$$

where $j \in \{1, ..., H - h + 1\}$ and $i \in \{1, ..., W - w + 1\}$.

As for the one-dimensional convolution, computing this corresponds to do at each location, the element-wise product between the filter and the input elements of the tensor that the filter overlaps, and then, to sum up the results obtained for each location. The following example gives a better understanding of what happens.

Example B.2. To simplify the situation, suppose that $C = 1$, i.e. we are working with 2D tensors. If

$$\mathbf{X} = \begin{pmatrix} 2 & 3 & 4 & 5 \\ -1 & 1 & 0 & 1 \\ 0 & 3 & 1 & 2 \\ 1 & -3 & -4 & 2 \end{pmatrix}$$

and

$$\mathbf{U} = \begin{pmatrix} 6 & -1 & 0 \\ -2 & 1 & 2 \end{pmatrix},$$

we have $H = W = 4$, $h = 2$ and $w = 3$. The convolution $\mathbf{X} * \mathbf{U}$ produces a tensor of size $(H - h + 1) \times (W - w + 1) = (4 - 2 + 1) \times (4 - 3 + 1) = 3 \times 2$. Let us compute the elements of this output tensor. We have

$$\begin{aligned}
(\mathbf{X} * \mathbf{U})[1, 1] &= \sum_{n=1}^{2} \sum_{m=1}^{3} \mathbf{X}_{n+0,\, m+0}\ \mathbf{U}_{n,\, m} \\
&= \mathbf{X}_{1,1}\mathbf{U}_{1,1} + \mathbf{X}_{1,2}\mathbf{U}_{1,2} + \mathbf{X}_{1,3}\mathbf{U}_{1,3} + \mathbf{X}_{2,1}\mathbf{U}_{2,1} + \mathbf{X}_{2,2}u_{2,2} + \mathbf{X}_{2,3}\mathbf{U}_{2,3} \\
&= 2.6 + 3.(-1) + 4.0 + (-1).(-2) + 1.1 + 0.2 \\
&= 12 - 3 + 2 + 1 \\
&= 12.
\end{aligned}$$

This is equivalent to computing the following element-wise product

$$\begin{pmatrix} 2 & 3 & 4 \\ -1 & 1 & 0 \end{pmatrix} \odot \begin{pmatrix} 6 & -1 & 0 \\ -2 & 1 & 2 \end{pmatrix} = \begin{pmatrix} 12 & -3 & 0 \\ 2 & 1 & 0 \end{pmatrix}$$

and then, to sum up all elements of the result. In this product, the first term is a sub-tensor of $\mathbf{X}$ and the second one is the filter $\mathbf{U}$. This provides 12 as found previously. Let us compute the value of $(\mathbf{X} * \mathbf{U})[1, 2]$. We have

$$\begin{aligned}
(\mathbf{X} * \mathbf{U})[1, 2] &= \sum_{n=1}^{2} \sum_{m=1}^{3} \mathbf{X}_{n+0,\, m+1}\; \mathbf{U}_{n,\, m} \\
&= \mathbf{X}_{1,2}\mathbf{U}_{1,1} + \mathbf{X}_{1,3}\mathbf{U}_{1,2} + \mathbf{X}_{1,4}\mathbf{U}_{1,3} + \mathbf{X}_{2,2}\mathbf{U}_{2,1} + \mathbf{X}_{2,3}\mathbf{U}_{2,2} + \mathbf{X}_{2,4}\mathbf{U}_{2,3} \\
&= 3.6 + 4.(-1) + 5.0 + 1.(-2) + 0.1 + 1.2 \\
&= 18 - 4 - 2 + 2 \\
&= 14.
\end{aligned}$$

This is equivalent to computing the following element-wise product between a sub-tensor of $\mathbf{X}$ and the kernel $\mathbf{U}$

$$\begin{pmatrix} 3 & 4 & 5 \\ 1 & 0 & 1 \end{pmatrix} \odot \begin{pmatrix} 6 & -1 & 0 \\ -2 & 1 & 2 \end{pmatrix} = \begin{pmatrix} 18 & -4 & 0 \\ -2 & 0 & 2 \end{pmatrix}$$

and then, to sum up all elements of the obtained 2D-tensor. This provides 14 as found previously. Doing the same for the other tensor values, we get the final result

$$\mathbf{X} * \mathbf{U} = \begin{pmatrix} 12 & 14 \\ -2 & 5 \\ -16 & 23 \end{pmatrix}.$$

Notice that for each value of the output tensor, the element-wise product is computed between the kernel and a tensor of the same size that slides on the width and height of the input tensor $\mathbf{X}$.

Now that we have seen how it works for 2D-tensors, we illustrate through Figure 1 how the output tensor is computed for a 3D input tensor. This figure illustrates the idea that the filter slides through the input tensor on the width first, see 1), 2) and 3), and produces an output value for each location. Sliding the filter over the width of the input tensor corresponds to computing the values of the first row of the output tensor. Then, from 4) the filter has slid over the height and continues to slide over the width. The process goes on, see 5): the filter slides over the width and height. At the end, we obtain 6) where the output tensor is of size $(H - h + 1) \times (W - w + 1)$ as explained before.

Remark B.2. As previously mentioned, it is possible to apply D convolutions in the same way and produce an output tensor of size $D \times (H - h + 1) \times (W - w + 1)$, where D corresponds to the depth. In this case, we obtain something visually similar to Figure 2.

To conclude about convolutional layers, it is important to remember that these layers use filters which slide on the width and the height of images and produce new images with modified features. In practice, the filter will correspond to the weight matrix, which contains all parameters of the network.

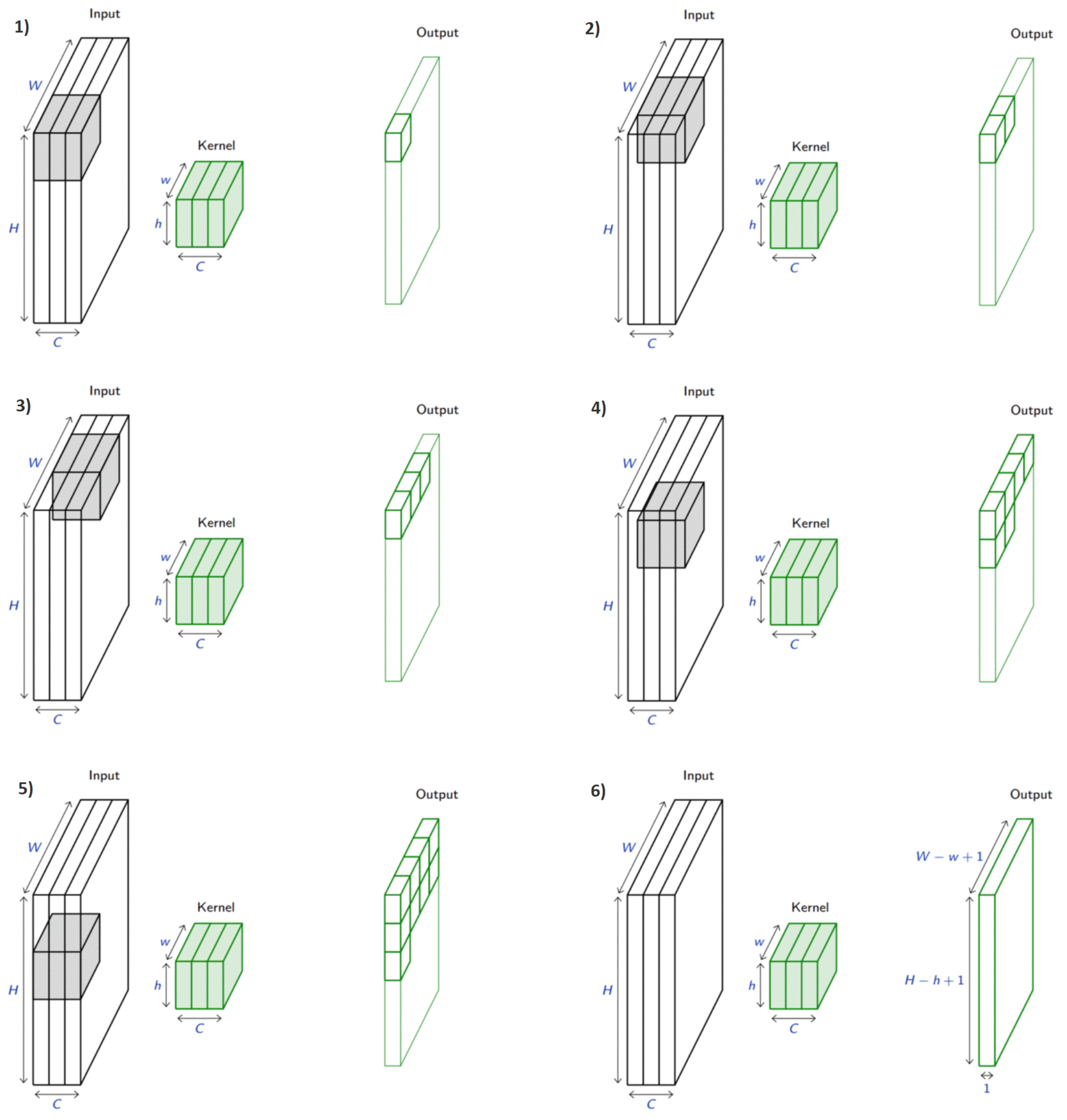


Figure 1: Illustration of how a filter (or kernel) slides through an input tensor in order to compute a convolution. © [37]

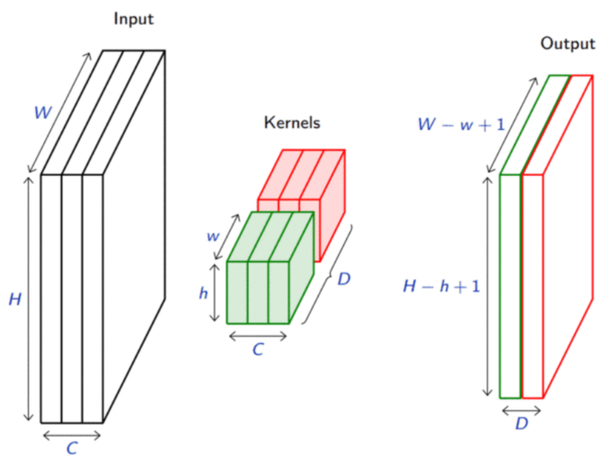


Figure 2: Illustration of the application of D convolutions, where D is the depth. © [37]

B.2 Pooling layer

Pooling layers have already been introduced in Section 2.2.5.2. The pooling operation, whose goal is to reduce the size of the input while preserving its whole structure, was defined as follows.

Definition B.3. Consider a pooling area $h \times w$ and a 3D input tensor $\mathbf{X} \in \mathbb{R}^{C\times(rh)\times(sw)}$ with h, w, C, r and $s \in \mathbb{N}$. The **pooling** operation produces an output tensor $\mathbf{O} \in \mathbb{R}^{C\times r\times s}$.

- If the **maximum pooling** or **max-pooling** is used, the output tensor is given by

$$\mathbf{O}_{c,j,i} = \max_{n<h,\, m<w} \mathbf{X}_{c,\, r(j-1)+n,\, s(i-1)+m}$$

  where $c \in \{1, ..., C\}$, $j \in \{1, ..., r\}$ and $i \in \{1, ..., s\}$.

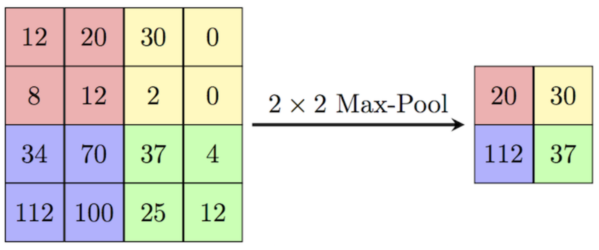
- If the **average pooling** is used, the output tensor is given by

$$\mathbf{O}_{c,j,i} = \frac{1}{hw} \sum_{n=1}^{h} \sum_{m=1}^{w} \mathbf{X}_{c,\, r(j-1)+n,\, s(i-1)+m}$$

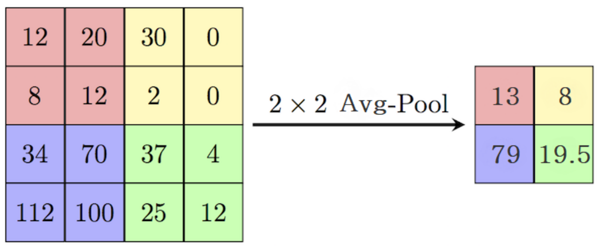
  where $c \in \{1, ..., C\}$, $j \in \{1, ..., r\}$ and $i \in \{1, ..., s\}$.

As the definition may not be very intuitive, the following example is intended to illustrate it.

Example B.3. Figure 3 (a) illustrates max-pooling on a 4×4 tensor using a 2×2 pooling area. In other terms, it replaces a set of values by the maximum. In the case of images, these values are values at pixels. Figure 3 (b) illustrates average pooling on the same 4×4 tensor and also uses a 2×2 pooling area. In the case of an image, a set of pixels is replaced by their average.



(a)



(b)

Figure 3: Illustrations of 2×2 pooling operations on an input tensor of size 4×4. (a) 2×2 max-pooling operation. The maximum is selected in each of the four coloured blocks to get the values of the output tensor. (b) 2×2 average pooling operation. The average is taken over each of the four coloured blocks to get the values of the output tensor. © [35]

In practice, when a max-pooling layer is applied, the scale of the image is reduced by removing the least relevant information and retaining the main features. Conversely, when an average pooling layer is used, more information is preserved, but the main features may be diluted. This was already explained and illustrated in Example 2.2.2.

Finally, Figure 4 illustrates how the pooling area moves through the 3D input tensor to compute the final value of each element of the output tensor.

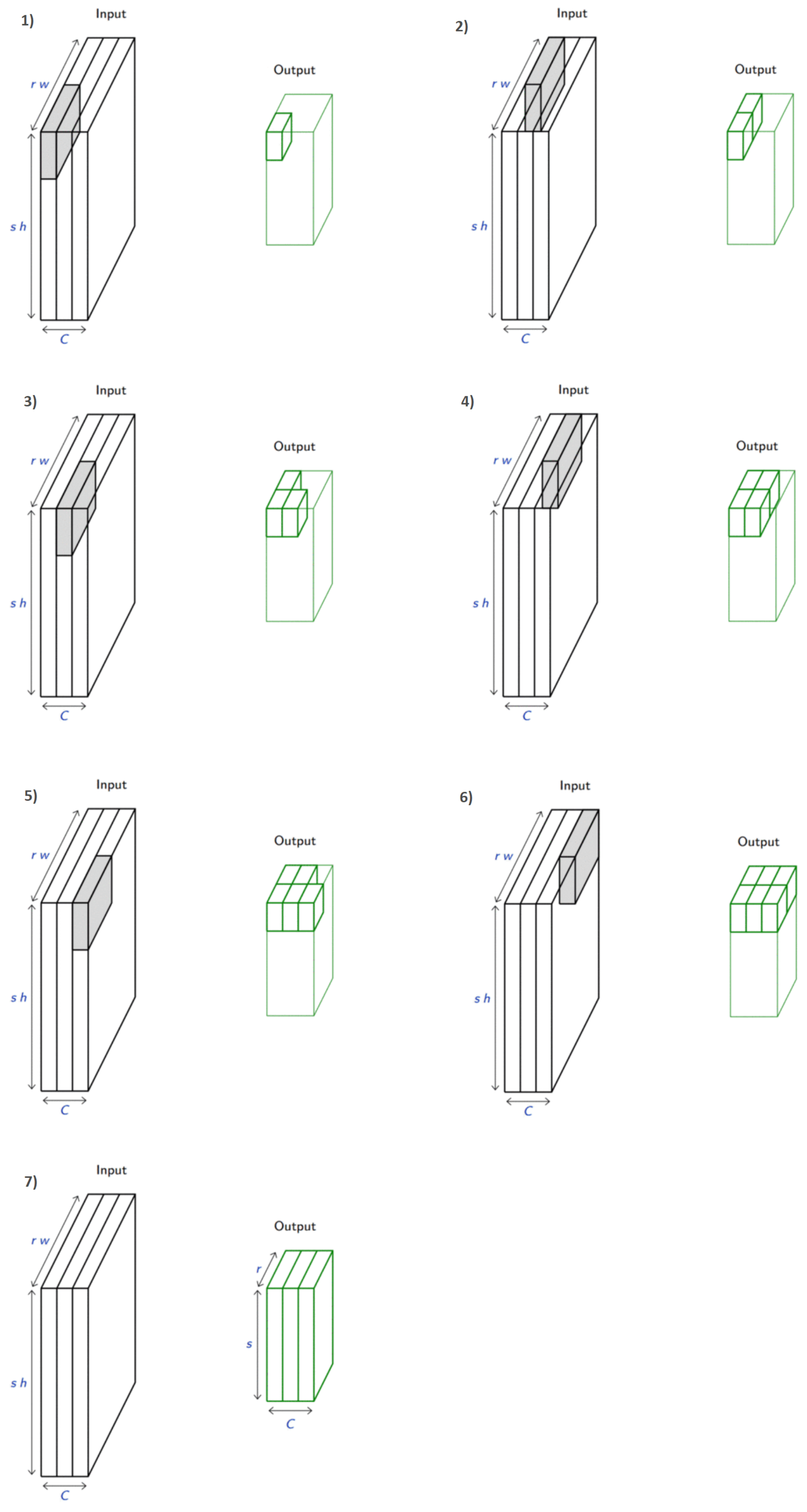


Figure 4: Illustration of how the pooling area (grey rectangle) moves through the 3D input tensor to compute the final value of each element of the output tensor. © [37]

C Local hyperparameters study

The study of the local hyperparameters of YOLOv4 was not presented in Section 4.3.1 because it was very similar to the one of YOLOv3. This section explains how we studied their influence in order to decide how to set them. The local hyperparameters are the learning rate, momentum, decay and steps-scales. Each test of this section was done on P dataset using 7500 iterations.

As for YOLOv3, for each local hyperparameter, several values were tested and four tests for each of them were performed. The resulting performance measures (F1-score, precision, recall and mAP) were averaged and then reported in a table for the validation and test sets. The validation set is used for the hyperparameter tuning, while the values obtained on the test set are displayed for information and thus are not analysed.

Learning rate

Following values of the learning rate were tested: 0.0005, 0.001 (default value), 0.0025 and 0.005. The table below contains the averaged performance measures for each of these values.

Learning rate	**Validation set**				**Test set**			
	F1-score	*precision*	*recall*	*mAP*	*F1-score*	*precision*	*recall*	*mAP*
0.0005	**0.845**	**0.875**	0.815	0.684	0.765	0.875	0.682	0.600
0.001	0.842	0.860	**0.822**	0.682	0.775	0.862	**0.707**	0.584
0.0025	0.842	0.867	0.820	**0.690**	**0.782**	0.880	0.705	0.606
0.005	0.832	0.865	0.795	0.688	0.757	**0.882**	0.665	**0.618**

The learning rate is set to 0.0025 because, on the validation set, it leads to slightly better results for the mAP than the other learning rate values, which is rather poor compared to the F1-score. If we consider that a batch size of 32 is relatively small, this is not very consistent with theory because a small learning rate should lead to better performance and the selected learning rate value is not so small.

Momentum

The values 0.7, 0.8, 0.9 and 0.949 (default value) were tested for momentum. The table below contains the averaged performance measures for each of these values.

Momentum	**Validation set**				**Test set**			
	F1-score	*precision*	*recall*	*mAP*	*F1-score*	*precision*	*recall*	*mAP*
0.7	0.832	0.865	0.807	0.686	0.765	0.870	0.682	0.603
0.8	**0.845**	0.862	**0.827**	0.688	0.767	0.870	0.690	0.605
0.9	0.840	0.865	0.815	0.689	0.770	0.872	0.690	0.595
0.949	0.842	**0.867**	0.820	**0.690**	**0.782**	**0.880**	**0.705**	**0.606**

The momentum value is set to its default value, 0.949, because it leads to the highest value of the mAP on the validation set. According to theory, this term should be greater than the learning rate, which is indeed the case.

Decay

Following values were tested for the decay: 0.00005, 0.0005 (default value), 0.005 and 0.05. The table below contains the averaged performance measures for each of these values.

Decay	**Validation set**				**Test set**			
	F1-score	*precision*	*recall*	*mAP*	*F1-score*	*precision*	*recall*	*mAP*
0.00005	0.837	0.870	0.807	0.684	0.775	0.877	0.697	0.593
0.0005	**0.842**	0.867	**0.820**	**0.690**	**0.782**	0.880	**0.705**	0.606
0.005	0.827	0.855	0.802	0.689	0.760	0.860	0.677	**0.619**
0.05	0.717	**0.895**	0.597	0.556	0.595	**0.892**	0.445	0.434

The decay value is set to its default value, 0.0005, because it leads to the best F1-score and mAP on the validation set. As theoretically expected, a small decay leads to better performance measures and is more appropriate with a complex dataset, which is our case.

Steps-Scales

The values tested for steps are 6000-6750 and 5000-6500, and those for scales are 0.1-0.1 and 0.5-0.2. The four combinations of values were tested. The table below contains the averaged performance measures for each steps-scales value mentioned above.

Steps	**Scales**	**Validation set**				**Test set**			
		F1-score	*precision*	*recall*	*mAP*	*F1-score*	*precision*	*recall*	*mAP*
6000-6750	**0.1-0.1**	**0.842**	0.867	0.820	0.690	**0.782**	0.880	0.705	**0.606**
6000-6750	0.5-0.2	0.837	**0.877**	0.800	**0.695**	0.780	**0.890**	0.692	0.592
5000-6500	0.1-0.1	0.840	0.875	0.810	0.690	0.770	0.872	0.690	**0.606**
5000-6500	0.5-0.2	0.837	0.847	**0.830**	0.686	0.772	0.845	**0.710**	0.571

As the differences between the F1-score and the mAP are small, the steps-scales hyperparameter is kept to its default value, which is 6000-6750 with 0.1-0.1.

Summary

Based on the values selected for the global hyperparameters in Section 4.3.1, the following table summarises how to set each hyperparameter value of YOLOv4.

Hyperparameter	**Value**
Image size	608×608
Batch size	32
Subdivisions	32
Learning rate	0.0025
Burn in	1000
Momentum	0.949
Decay	0.0005
Steps-Scales	6000-6750 with 0.1-0.1

Bibliography

[1] AGARWAL, Devraj. *A Review of the Math Used in Training a Neural Network.* 2020. Available from <https://levelup.gitconnected.com/a-review-of-the-math-used-in-training-a-neural-network-9b9d5838f272> (visited on September 23, 2022).

[2] ALASKA SATELLITE FACILITY (ASF). Available from <https://asf.alaska.edu/>.

[3] ALMOG, Uri. *YOLO V3 Explained.* 2020. Available from <https://towardsdatascience.com/yolo-v3-explained-ff5b850390f> (visited on August 27, 2022).

[4] AMRUTHNATH, Nagdev. *Why balancing your data set is important?* 2020. Available from <https://www.r-bloggers.com/2020/06/why-balancing-your-data-set-is-important/> (visited on July 20, 2023).

[5] ASHWIN, Anthony, Peter CHAZHOOR, Edmond SHU-LIM HO, Bin GAO and Wai LOK WOO. Deep transfer learning benchmark for plastic waste classification. 2022. Available from <https://intellrobot.com/article/view/4550> (visited on November 10, 2022).

[6] BASTIEN. *Introduction au machine learning – Partie 1/3 – Histoire.* 2019. Available from <https://blog.clevy.io/nlp-et-ia/introduction-machine-learning-1-3-histoire/> (visited on August 28, 2022).

[7] BIERHANCE, Thomas. *Do Batch Sizes Actually Need to be Powers of 2?: Is the fixation on powers of 2 for efficient GPU utilization an urban myth? Let's find out.* 2022. Available from <https://wandb.ai/datenzauberai/Batch-Size-Testing/reports/Do-Batch-Sizes-Actually-Need-to-be-Powers-of-2---VmlldzoyMDkwNDQx> (visited on December 4, 2022).

[8] BOCHKOVSKIY, Alexey, Chien-Yao WANG and Hong-Yuan MARK LIAO. Yolov4: optimal speed and accuracy of object detection. 2020. Available from <https://arxiv.org/pdf/2004.10934.pdf> (visited on August 27, 2022).

[9] BOCHKOVSKIY, Alexey, Chien-yao WANG and Hong-Yuan MARK LIAO. *YOLOv4: Optimal Speed and Accuracy of Object Detection.* 2020. Available from arXiv: 2004.10934 [cs.CV] (visited on July 20, 2023).

[10] BRUNNER, Karl-Heinz. *Espace et sécurité: le rôle de l'OTAN.* 2021-12-01. Assemblée parlementaire de l'OTAN. Organisation du traité de l'Atlantique nord (OTAN). Available from <https://www.nato-pa.int/download-file?filename=/sites/default/files/2021-12/025%20STC%2021%20F%20rev.%202%20fin%20-%20ESPACE%20ET%20SECURITE%20-%20BRUNNER%20.pdf>.

[11] CAMACHO, Cezanne. *Convolutional Neural Networks.* 2018. Available from <https://cezannec.github.io/Convolutional_Neural_Networks/> (visited on September 29, 2022).

[12] CAPPART, Alice. *Ship detection from SAR imaging and machine learning techniques.* MA thesis. University of Liège. 2023.

[13] CHAKRABORTY, Ananya. *How to Learn Mathematics For Machine Learning? What Concepts do You Need to Master in Data Science?* 2021. Available from <https://www.analyticsvidhya.com/blog/2021/06/how-to-learn-mathematics-for-machine-learning-what-concepts-do-you-need-to-master-in-data-science/> (visited on August 28, 2022).

[14] CHEN, Jiayao, Mingliang ZHOU, Dongming ZHANG, Hongwei HUANG and Fengshou ZHANG. Quantification of water inflow in rock tunnel faces via convolutional neural network approach. *Elsevier.* 2020. Available from <https://www.researchgate.net/publication/348078198_Quantification_of_water_inflow_in_rock_tunnel_faces_via_convolutional_neural_network_approach> (visited on November 10, 2022).

[15] EUROPEAN SPACE AGENCY (ESA). *Level-1 SLC Products.* Available from <https://sentinels.copernicus.eu/web/sentinel/technical-guides/sentinel-1-sar/products-algorithms/level-1-algorithms/single-look-complex> (visited on August 6, 2023).

[16] EUROPEAN SPACE AGENCY (ESA). *Sentinel Online: Orbit.* Available from <https://sentinels.copernicus.eu/web/sentinel/missions/sentinel-1/satellite-description/orbit> (visited on April 10, 2023).

[17] FERGUSON, Max, Ronay AK, Yung-Tsun LEE and Kincho LAW. Automatic localization of casting defects with convolutional neural networks. *In*: 2017, pp. 1726–1735. Available from <https://www.researchgate.net/publication/322512435_Automatic_localization_of_casting_defects_with_convolutional_neural_networks>.

[18] FLEURET, François. *Deep learning course.* University of Geneva, 2022. Available from <https://fleuret.org/dlc/#lectures>.

[19] GAD, Ahmed. *A Comprehensive Guide to the Backpropagation Algorithm in Neural Networks.* 2022. Available from <https://neptune.ai/blog/backpropagation-algorithm-in-neural-networks-guide> (visited on November 6, 2022).

[20] GAO, Hao. *Object Localization in Overfeat.* 2017. Available from <https://towardsdatascience.com/object-localization-in-overfeat-5bb2f7328b62> (visited on November 24, 2022).

[21] GEURTS, Pierre and Louis WEHENKEL. *Introduction to machine learning.* University of Liège, 2021. Available from <https://people.montefiore.uliege.be/lwh/AIA/>.

[22] GIRSHICK, Ross. Fast R-CNN. 2015. Available from <https://arxiv.org/pdf/1504.08083.pdf> (visited on September 18, 2022).

[23] GIRSHICK, Ross, Jeff DONAHUE, Trevor DARRELL and Jitendra MALIK. Rich feature hierarchies for accurate object detection and semantic segmentation. 2014. Available from <https://arxiv.org/pdf/1311.2524.pdf> (visited on September 20, 2022).

[24] GLADCHUK, Veronika. *The History of Machine Learning: How Did It All Start?* 2020. Available from <https://labelyourdata.com/articles/history-of-machine-learning-how-did-it-all-start> (visited on August 28, 2022).

[25] GOODFELLOW, Ian, Yoshua BENGIO and Aaron COURVILLE. *Deep Learning.* MIT Press, 2016. http://www.deeplearningbook.org.

[26] HAESBROECK, Gentiane. *Modèles linéaires.* University of Liège, 2023.

[27] HASTIE, Trevor, Robert TIBSHIRANI and Jerome FRIEDMAN. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer Series in Statistics, 2009.

[28] HE, Kaiming, Georgia GKIOXARI, Piotr DOLLÀR and Ross GIRSHICK. Mask R-CNN. 2018. Available from <https://arxiv.org/pdf/1703.06870.pdf> (visited on September 4, 2022).

[29] HE, Kaiming, Xiangyu ZHANG, Shaoqing REN and Jian SUN. Deep residual learning for image recognition. *In*: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 770–778. Available from DOI: 10.1109/CVPR.2016.90.

[30] HVILSHØJ, Frederik. *Introduction to Balanced and Imbalanced Datasets in Machine Learning.* 2022. Available from <https://encord.com/blog/an-introduction-to-balanced-and-imbalanced-datasets-in-machine-learning/> (visited on July 20, 2023).

[31] *ImageNet.* Available from <https://www.image-net.org/>.

[32] JAISWAL, Sonoo. *Essential Mathematics for Machine Learning | Important concepts of Mathematics for Machine Learning.* Available from <https://www.javatpoint.com/essential-mathematics-for-machine-learning> (visited on August 28, 2022).

[33] KIRKOVE, Murielle. SYMPA – Vessel detection state of the art. *CSL.* 2016.

[34] *LabelImg.* Available from <https://github.com/HumanSignal/labelImg>.

[35] LI, Fei-Fei, Yunzhu LI and Ruohan GAO. *CS231n: Deep Learning for Computer Vision.* Stanford University, 2023. Available from <http://cs231n.stanford.edu/index.html>.

[36] LI, Jianwei, Congan XU, Hang SU, Long GAO and Taoyang WANG. Deep learning for sar ship detection: past, present and future. *Remote Sensing.* 2022, 14 (11). ISSN 2072-4292. Available from <https://www.mdpi.com/2072-4292/14/11/2712>.

[37] LOUPPE, Gilles. *Deep learning.* University of Liège, 2022. Available from <https://github.com/glouppe/info8010-deep-learning>.

[38] MAEDA-GUTIÉRREZ, Valeria, Carlos E. GALVÁN-TEJADA and Laura A. ZANELLA-CALZADA. Comparison of convolutional neural network architectures for classification of tomato plant diseases. *Applied sciences.* 2020, 10 (4), pp. 1245–. ISSN 2076-3417. Available from <https://www.mdpi.com/2076-3417/10/4/1245/htm> (visited on October 1, 2022).

[39] MURPHY, Kevin Patrick. *Probabilistic Machine Learning: An introduction.* MIT Press, 2022.

[40] NAN, Mo and Yan LI. Improved faster RCNN based on feature amplification and oversampling data augmentation for oriented vehicle detection in aerial images. *Remote Sensing.* 2020, 12 (16). ISSN 2072-4292. Available from <https://www.mdpi.com/2072-4292/12/16/2558>.

[41] NATIONAL AERONAUTICS AND SPACE ADMINISTRATION (NASA). *What is Synthetic Aperture Radar?: Background information on synthetic aperture radar, with details on wavelength and frequency, polarization, scattering mechanisms, and interferometry.* Available from <https://www.earthdata.nasa.gov/learn/backgrounders/what-is-sar> (visited on August 6, 2023).

[42] PADILLA, Rafael, Sergio NETTO and Eduardo da SILVA. A survey on performance metrics for object-detection algorithms. *In*: 2020. Available from <https://www.researchgate.net/publication/343194514_A_Survey_on_Performance_Metrics_for_Object-Detection_Algorithms>.

[43] PYTORCH. *TorchVision Object Detection Finetuning Tutorial.* Available from <https://pytorch.org/tutorials/intermediate/torchvision_tutorial.html>.

[44] RASCHKA, Sebastian. *No, We Don't Have to Choose Batch Sizes As Powers Of 2.* 2022. Available from <https://sebastianraschka.com/blog/2022/batch-size-2.html> (visited on December 4, 2022).

[45] REDMON, Joseph. *Darknet: Open Source Neural Networks in C.* 2013-2016. Available from <https://pjreddie.com/darknet/>.

[46] REDMON, Joseph, Santosh DIVVALA, Ross GIRSHICK and Ali FARHADI. You only look once: unified, real-time object detection. 2016. Available from <https://arxiv.org/pdf/1506.02640.pdf> (visited on August 27, 2022).

[47] REDMON, Joseph and Ali FARHADI. Yolo9000: better, faster, stronger. 2016. Available from <https://openaccess.thecvf.com/content_cvpr_2017/papers/Redmon_YOLO9000_Better_Faster_CVPR_2017_paper.pdf> (visited on August 27, 2022).

[48] REDMON, Joseph and Ali FARHADI. Yolov3: an incremental improvement. 2018. Available from <https://arxiv.org/pdf/1804.02767.pdf> (visited on August 27, 2022).

[49] REN, Shaoqing, Kaiming HE, Ross GIRSHICK and Jian SUN. Faster R-CNN: towards real-time object detection with region proposal networks. 2016. Available from <https://arxiv.org/pdf/1506.01497.pdf>.

[50] ROSEBROCK, Adrian. *Intersection over Union (IoU) for object detection.* 2016. Available from <https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/> (visited on August 28, 2022).

[51] SAS, Institute. *Object Detection: Faster R-CNN.* 2021. Available from <http://go.documentation.sas.com/doc/en/pgmcdc/8.11/casdlpg/p1np8zbnoyd0brn1dhehthuuxj4q.htmn0h8og63jxwg5ln11q3vx692w9ww> (visited on November 11, 2022).

[52] SERMANET, Pierre, David EIGEN, Xiang ZHANG, Michael MATHIEU, Rob FERGUS and Yann LECUN. Overfeat: integrated recognition, localization and detection using convolutional networks. 2014. Available from <https://arxiv.org/pdf/1312.6229.pdf> (visited on August 8, 2023).

[53] SHRESTHA, Sandesh. *Detection of Human Presence and Status of Electric Appliancesfor Home Automation and Human Trackingusing Video Surveillance.* MA thesis. University of Tribhuvan. Available from <http://ise.ait.ac.th/wp-content/uploads/sites/57/2020/12/Detection-of-Human-Presence-and-Status-of-Electric-Appliances-for-Home-Automation-and-Human-Tracking-using-Video-Surveillance.pdf>. 2019.

[54] SIMONYAN, Karen and Andrew ZISSERMAN. Very deep convolutional networks for large-scale image recognition. 2014. Available from <https://arxiv.org/pdf/1409.1556.pdf>.

[55] SMITH, Leslie N. A disciplined approach to neural network hyper-parameters: part 1 – learning rate, batch size, momentum, and weight decay. 2018 . Available from arXiv: 1803.09820 [cs.LG] (visited on July 27, 2023).

[56] STOCK, Damien. *Automatic Oil Spill detection and monitoring with supervised machine learning and SAR remote sensing.* MA thesis. University of Liège. 2020.

[57] TURING, Alan Mathison. Computing machinery and intelligence. *Mind.* 1950, 59 (236), pp. 433–460. Available from <http://www.jstor.org/stable/2251299> (visited on August 28, 2022).

[58] WIKIPEDIA. *Timeline of machine learning.* Available from <https://en.wikipedia.org/wiki/Timeline_of_machine_learning> (visited on August 28, 2022).

[59] XIE, Saining, Ross GIRSHICK, Piotr DOLLÁR, Zhuowen TU and Kaiming HE. Aggregated residual transformations for deep neural networks. 2017. Available from <https://arxiv.org/pdf/1611.05431v2.pdf>.

[60] ZHANG, Aston, Zachary C. LIPTON, Mu LI and Alexander J. SMOLA. *Dive into dep learning.* 2023. Available from <https://d2l.ai/d2l-en.pdf>.

[61] ZHANG, Tianwen, Xiaoling ZHANG, Xiao KE, Xu ZHAN, Jun SHI, Shunjun WEI, Dece PAN, Jianwei LI, Hao SU, Yue ZHOU and Durga KUMAR. LS-SSDD-v1.0: a deep learning dataset dedicated to small ship detection from large-scale sentinel-1 sar images. *Remote Sensing.* 2020.

[62] ZHANG, Tianwen, Xiaoling ZHANG, Jianwei LI, Xiaowo XU, Baoyou WANG, Xu ZHAN, Yanqin XU, Xiao KE, Tianjiao ZENG, Hao SU, Israr AHMAD, Dece PAN, Chang LIU, Yue ZHOU, Jun SHI and Shunjun WEI. Sar ship detection dataset (SSDD): official release and comprehensive data analysis. *Remote Sensing.* 2021, 13 (18). ISSN 2072-4292. Available from <https://www.mdpi.com/2072-4292/13/18/3690>.