TÉCNICO
LISBOA

Deep Learning for automatic target recognition in synthetic aperture radar images

Nuno Manuel de Barros Ferreira

Thesis to obtain the Master of Science Degree in

Aerospace Engineering

Supervisor: Prof. Maria Margarida Campos da Silveira

Examination Committee

Chairperson: Prof. José Fernando Alves da Silva
Supervisor: Prof. Maria Margarida Campos da Silveira
Member of the Committee: Prof. Pedro Miguel Berardo Duarte Pina

October 2020

Dedicated to my grandparents

Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

A hundred paragraphs would not suffice to describe my feelings for those who helped and guided me through this incredible journey to become an engineer.

Firstly, I would like to thank my supervisor, professor Margarida Silveira, who not once failed to give me guidance and support throughout the last year. All the work presented here, from theory to practice, would undoubtedly be impossible to achieve without her help and supervision. Thank you for always pushing me to do my best.

To my oldest friends from Penafiel who, despite the distance that separated us in these years of college, never stopped supporting me. In particular, to Zé, Jimbas, Zouev, João, and Isa, who have always been there when I needed it the most, showing me what true friendship really means.

To Francisco and Roque, my big thank you for always having my back during the hardest times of the semesters. I could not have wished for better classmates, who rapidly became two of my closest friends.

A deep thank you to the MyNutriScan's team. We have had a truly amazing journey together. In particular to Nuno: a colleague, roommate, and dear friend, who never ceased to support me.

To all my family, who always believed in me. Particularly, to my uncle David, aunt Fernanda, and my godfather, Avelino, who had a tremendous impact in my growth as a person.

To my grandparents, José and Idalina, for always being by my side when I was growing up. This thesis is dedicated to you.

To my brother, to whom I always looked up to when I was a kid.

To my girlfriend and dearest friend, Marta. It is impossible to put into words how thankful I am for having such an amazing person in my life. I am truly indebted to you. Thank you for your unconditional support during the hardest times. For everything.

Finally, thank you mom and dad, for providing me with absolutely everything I ever needed, even through the most difficult times. All my life achievements are a result of your hard work, and the education you gave me. To make you proud has always been my number one priority.

Resumo

O reconhecimento automático de alvos em imagens de satélite é uma tarefa fundamental na vigilância marítima. Anualmente, milhares de refugiados perdem a vida no mar; a pesca ilegal contribui significativamente para sobreexploração dos recursos marinhos, que leva ao desiquilíbrio de ecossistemas; além disso, outras atividades, tais como o tráfico de armas e drogas, são levadas a cabo pelo mar. Portanto, é essencial haver uma boa monitorização das águas, de modo a mitigar estes problemas. Tipicamente usam-se técnicas de aprendizagem automática supervisionadas para realizar tarefas relacionadas com a deteção de objetos, recorrendo a etiquetas durante o treino dos algoritmos. No entanto, o processo de *labelling* pode ser moroso e dispendioso. Técnicas de aprendizagem não supervisionadas, por outro lado, possibilitam a extração de informação relevante dos dados sem recorrer ao uso de etiquetas, sendo particularmente aliciantes na análise de imagens de satélite, dada a enorme disponibilidade de dados existente, e o seu constante aumento. Neste trabalho foi abordado o estudo da possibilidade da deteção automática de anomalias em imagens de radar de abertura sintética (SAR), utilizando *Variational Autoencoders* (VAEs), de uma forma não supervisionada. No modelo proposto, o *encoder* do VAE é treinado para mapear imagens normais num espaço latente. Para se realizar a classificação de um conjunto de imagens, com e sem anomalias, estas são codificadas pelo *encoder* num espaço latente, onde as imagens normais (sem anomalias) ficam aglomeradas, e as imagens anómalas dispersas no espaço. Finalmente, um algoritmo de *clustering* é aplicado ao espaço gerado, permitindo identificar as imagens anómalas.

Palavras-chave: Aprendizagem profunda, *Variational Autoencoder*, Deteção de Anomalias, Aprendizagem não supervisionada, SAR

Abstract

The automatic target recognition (ATR) in satellite images is an important application in maritime surveillance. Annually, thousands of refugees lose their lives in the sea; illegal fishing highly contributes to the over-exploitation of fish resources, perturbing ecosystems; besides, other activities, such as drug and firearms trafficking, are conducted through the sea. Hence, it is clear that a good maritime surveillance is key to mitigate these problems. Usually, object detection tasks are performed by supervised machine learning algorithms, making use of labels during training. However, the daunting process of labelling images is often expensive and time consuming.

On the other hand, unsupervised learning techniques allow the extraction of relevant information from data, without making use of labels, being particularly interesting for the analysis of satellite images, given the huge amount of available data, and its constant increase. This work addresses the possibility of automatically detect anomalies in synthetic aperture radar (SAR) images, using Variational Autoencoders (VAEs), in an unsupervised manner. In the proposed framework, the encoder of the VAE is trained to map normal images in a latent space. To perform classification of a given set of images, with and without anomalies, these are encoded in a latent space by the encoder, where normal images (without anomalies) are clustered together and the anomalous images are spread across the space. Finally, a clustering algorithm is applied to this generated space, being able to identify the anomalous images among the set.

Keywords: Deep learning, Variational autoencoders, Anomaly detection, Unsupervised learning, SAR

Contents

List of Tables

List of Figures

Nomenclature

1-D	One dimensional
2-D	Two dimensional
AE	Autoencoder
AKDEMO	Alaska SAR demonstration
ATR	Automatic target recognition
CA-CFAR	Cell-averaging CFAR
CFAR	Constant false alarm rate
CNN	Convolutional Neural Network
DCAE	Deep convolutional autoencoder
DLVM	Deep latent variable model
DNN	Deep neural network
DRENN	Deep recurrent encoding neural network
ELBO	Evidence lower bound
FC	Fully connected
FPN	Feature pyramid network
GLTR	Generalized-likelihood ratio test
GMM	Gaussian mixture model
GOCA-CFAR	Greatest of CA-CFAR
GPU	Graphics processing unit
HOG	Histogram of oriented gradients
iDPolRAD	Intensity dual-polarization ratio anomaly detector
IGARSS	International geoscience and remote sensing symposium

IUU	Illegal, unreported and unregulated
JRC	Joint research center
KDE	Kernel density estimation
KL	Kullback-Leibler
KNN	K-nearest neighbor
LSTM	Long-short term memory
MAP	Maximum *a aposteriori*
MLE	Maximum likelihood estimate
MR-SSD	Multi-resolution single shot detector
MSE	Mean squared error
MSTAR	Moving and stationary target acquisition and recognition
NFCAE	Non-negative and Fisher constrained autoencoder
OMW	Ocean monitoring workstation
OS-CFAR	Order Statistics CFAR
PCA	Principal component analysis
PD	Probability of detection
PDF	Probability density function
PFA	Probability of false alarm
PUT	Pixels under test
RCNN	Region-based convolutional neural network
RCS	Radar cross section
ReLU	Rectified linear unit
RoI	Region of interest
RPN	Region proposal network
SAE	Stacked autoencoder
SAR	Synthetic aperture radar
SGD	Stochastic gradient descent
SIFT	Scale invariant feature transform

SNR	Signal to noise ratio
SOCA-CFAR	Smallest of CA-CFAR
Soft-NMS	Soft-Non-maximum Suppression
SSAE	Stacked sparse autoencoder
SSD	Single shot detector
SSDD	SAR ship detection dataset
SVA	Spatially invariant apodization
SVM	Support vector machine
SWOS-CFAR	Switched order statistics CFAR
TPLBP	Three-Patch Local Binary Pattern
VAE	Variational autoencoder
XML	Extensible markup language
YOLO	You only look once

Chapter 1

Introduction

1.1 Motivation

Maritime surveillance is an important activity for many countries. From the early detection of oil spills to the identification of clandestine vessels, the relevance of an effective monitoring of the sea is evident. Annually, thousands of refugees lose their lives in the sea. According to [1], from 2015 to 2018, over 14000 deaths of migrants trying to reach Europe by crossing the Mediterranean were confirmed. Often travelling on inappropriate and overcrowded vessels, the smugglers who transport these people can abandon the boat and leave them to be found by international navies [2]; illegal, unreported and unregulated (IUU) fishing highly contributes to the overexploitation of fish resources, perturbing ecosystems and fish populations. The fraction of fish stocks within biologically sustainable levels in 2015 was 66.9%, meaning that 33.1% of fish stocks were fished at an unsustainable level, a number that has been following an increasing trend since 1975 [3], and, according to [4], there is a correspondence between the regional estimates of illegal and unreported fishing and the number of depleted stocks in those regions. IUU also contributes to the destruction of crucial components of the maritime ecosystem [5]. Several other illegal activities are conducted by sea, such as cocaine trafficking, which, according to the United Nations Office on Drugs and Crime, is trafficked to Europe mostly by sea. Oil spills are also a major concern. When an oil slick is detected, the responsible authorities must quickly activate appropriate protocols to control their damage and ecological impact.

It is self-evident that an adequate sea monitoring is crucial in the prevention of the aforementioned events and in the control of the impact of their consequences, and it is within this scope that remote sensing image scene classification - an active research topic in the field of aerial and satellite image analysis, that aims to categorize scene images into a set of meaningful classes, based on the image contents [6] - comes into play.

Synthetic Aperture Radar (SAR) is a form of radar, used to create images. It has some drawbacks such as being only suitable for stationary and slow moving targets, and being affected by speckle, a granular disturbance, usually modeled as a multiplicative noise, that affects SAR images and other coherent images. Nevertheless, SAR is regarded as one of the most suitable sensors for object detection and environment monitoring as it showcases a wide coverage range and important advantages when compared to other data acquisition methods commonly used in remote sensing, namely the ability to acquire data regardless of daylight and weather conditions, making the analysis of SAR images a very important field of research in remote sensing, more specifically in the field of automatic target recognition (ATR).

Traditional target recognition methods in SAR images are based on constant false alarm rate (CFAR) methods, which use a threshold to keep the false alarm rate constant [7]. In these methods, the sea clutter background is modeled according to a suitable distribution and a threshold is set to achieve an assigned probability of false alarm (PFA) [8]. One issue of CFAR algorithms is that they are not able to detect targets with intensity values close to the sea clutter and the threshold computation can represent a time-consuming procedure [8]. Moreover, these methods perform poorly in rough sea conditions. Variations of the CFAR algorithm have also been proposed, such as the bilateral CFAR algorithm [9], which uses a combination of the intensity distribution and the spatial distribution of the SAR images.

Other novel ship detection methods - such as a detector based on the generalized-likelihood ratio test (GLRT) [8] and the depolarization ratio anomaly detector using dual-polarization SAR images [10] [11] - have been proposed.

In accordance to the emerging paradigm toward data intensive science, machine learning techniques have become increasingly important. In particular, deep learning has proven to be both a major breakthrough and an extremely powerful tool in many fields, namely in the field of computer vision, where deep learning algorithms have managed to equal or even surpass human-level performance in tasks that computers used to struggle when trying to solve them. Object recognition, for instance, a seemingly effortless and rapid task for humans to perform, was very challenging for computers up until the appearance of deep neural networks (DNNs). Prior to the usage of such algorithms, traditional machine learning algorithms for object detection were roughly divided into region selections, such as scale-invariant feature transform (SIFT), and histogram of oriented gradients (HOG), and classifiers, such as support vector machines (SVMs). The beginning of the deep learning booming is often credited to the Image Net's image classification challenge winner in 2012, where a deep convolutional neural network called AlexNet surpassed all other contestants' algorithms, winning the challenge with impressively high accuracy and performance [12].

In the wake of the success of deep learning, in conjunction with the increasing availability of data, deep learning algorithms naturally started to take off in remote sensing. A series of supervised deep learning methods for ship detection as well as for ship classification have been proposed. Among the most successful methods are the you only look once v2 (YOLOv2) [13, 14], the faster region-based convolutional neural network (Faster RCNN) [15–17], and ResNet [18]. Despite the good results achieved by these methods, they are supervised methods which require large amounts of labelled data to be trained. Since the annotation process is time-consuming and requires domain knowledge from SAR experts, these methods do not take advantage of the huge and increasing amount of accessible SAR data. For instance, the Sentinel satellites, launched in 2014, had already collected about 25 PB of data, as of December 2017 [19].

It is clear that an algorithm able to process SAR images and perform target recognition regardless of annotation is of highly importance in order to take advantage of this incredible amount of data. And it is upon this reasoning that this thesis is developed.

1.2 Objectives

The goal of this thesis is to develop an unsupervised deep learning framework for anomaly detection in SAR ocean imagery, without resorting to image annotations during the training phase of the algorithm. The main idea is to learn image feature representations through a deep convolutional variational autoencoder, then followed by anomaly detection performed by a clustering algorithm.

It is expected that, after training, the developed model will be able to separate images with anomalies from images without anomalies (normal images), by learning key feature representations inherent to the images that comprise the training data.

1.3 Thesis outline and contributions

This document is comprised of six chapters. In the second chapter, an overview of the state of the art for target detection and representation learning in SAR images is provided, including literature review on some of the most common traditional methods, as well as more recent works. Given the vastness of the available literature, it is no easy task to make a comprehensive review. Nevertheless, we attempt to provide the reader a solid foundation around the topic, by touching the major key points in this matter.

In the third chapter, we dive into the theoretical and mathematical frameworks that are essential to understand and construct the main building blocks used to achieve the goal of this thesis. In addition, an explanation of how and why these building blocks are put together to build the proposed framework is provided.

The fourth chapter contains all the details of the implementation of the approach delineated in the previous chapter, including the steps to build and pre-process the dataset used to train the models, the specifications of the models, and the details of the training process.

Chapter five contains a detailed presentation and discussion of the results, with relevant figures and tables, so that the reader is able to understand how the framework performed during the whole process.

Finally, in chapter six, the main conclusions from this work are presented, along with the most relevant achievements, as well as possibilities of future developments.

Chapter 2

State-of-the-art

Modern SAR data acquisition systems can generate large amounts of data in a short period of time, verifying an obvious need for automatic target detection systems. In the context of sea ship monitoring, this is especially true, since much of the imagery contains only open ocean.

With the launch of the Seasat SAR system - a satellite SAR system specially designed to image the ocean - back in 1978, the possibility of imaging ships with spaceborne SAR systems was revealed and, since then, several other space-based SAR systems have been launched, reinforcing the need for this automatic tool. This led to a great interest in developing SAR ship detection systems, being the Ocean Monitoring Workstation (OMW), the Alaska SAR Demonstration (AKDEMO) system, the European Community Joint Research Center (JRC) system, and the Qinetiq's MaST system some examples of these systems [20].

The ongoing interest in SAR ship detection systems propelled a great amount of research around the topic, resulting in an extensive literature on algorithms designed to perform this task.

In this chapter we delve into the current state-of-the-art of target detection in SAR images, covering the modern techniques that present the best results for target detection in SAR imagery, as well as a brief overview of some traditional methods that have played an important role in this field of research. There are numerous strategies for implementing detection algorithms, evidenced by the vast literature around the topic. Different researchers tend to approach the problem of detecting targets from different perspectives, making it difficult to split the different methods into categories. Nonetheless, we divide the existing state-of-the-art methods into two distinct groups: traditional and modern, where the former category includes methods that have been used since the 90s, as well as their variations. The latter includes methods that make use of artificial intelligence algorithms to perform detection.

Afterwards, an overview of representation learning methods using autoencoders applied to SAR images in the literature is presented.

2.1 Target detection with traditional algorithms

Most of the traditional ocean target detection algorithms include some or all of the following stages: land masking, preprocessing, prescreening and discrimination. Hereby we briefly approach each of these stages, followed by some of the most used algorithms in this category.

2.1.1 Land masking

Land masking is the step where portions of land that may be present in the image to be processed are masked. This is an important step of the detection system, since the target detectors can produce high numbers of false alarms when applied to land areas, and dealing with false alarms can overtax the detector systems [7].

Land masking is commonly conducted through two methods: geographic registration and coastline detection.

Geographic registration consists of taking geographic maps and matching them with the given image. Although being one of the easiest approaches, this method is subject to errors. In the case of satellite imagery, the orbital parameters are not known precisely and registration errors can occur. Moreover, tidal variations can occur and small islands may not be marked on the maps. Hence, some land areas can be marked as ocean and vice-versa [7]. This method of land masking is applied to Radarsat data in [21] and to ERS-1 data in [22–24]. The initial step is to calculate geodetic longitude and latitude for the images obtained by the satellite, through its orbital parameters and other quantities [7]. Correlation techniques can be used to enhance the accuracy of this method from hundreds of meters to less then 100 meters [23, 24].

Automatic detection of coastlines overcomes the problems associated with the aforementioned geo-registration errors, the possible unavailability of satellite orbital data, inaccuracies in existing maps, shifting coastlines, and the missing mapping for small islands and rocks, as well as the tidal variations upon which the exposure of some rocks and shoals depend on [7]. One of the first SAR imagery target detection systems to mention this type of land masking was the Alenia Areospaczio ship detection system [25]. The used algorithm is based on [26], and the main steps involved are: filtering to remove speckle, application of an edge operator, edge map dilation with a mean filter, thresholding of the edge map histogram, and lastly applying a contour following algorithm. A terrain minimum area threshold is included to avoid masking ships. Another kind of segmentation approach is based on mature mathematical theory. Markovian random field is applied to detect the coastlines in [27], comparing the pixels to their surroundings and generating a rough coastline. Over the years, many other automatic coastline detection algorithms have been proposed [28–31].

2.1.2 Preprocessing

The preprocessing step has the objective to make latter stages of detection easier. This is a step that greatly depends on the methods used on the subsequent steps of the detection algorithm. For instance, although the despeckling process - a process of filtering the speckle noise in SAR images - enhances the visual appearance of a SAR image, there is no guarantee that it will facilitate the achievement of good target detection, unless the detection algorithm is designed to take into account this preprocessing step [7]. This step of speckle filtering for target detection is mentioned in [22, 32–35].

There are other preprocessing algorithms, such as image normalization, used in [36], which consists of dividing each pixel value by its local mean, calculated over a window of 30x30 pixels, with the intention to remove large image structures while retaining small local features. This step is used in a common detector algorithm, the cell-averaging CFAR (CA-CFAR) detector, which will be analysed in more detail later in this chapter.

Spatially variant apodization (SVA), a non-linear technique to suppress SAR processor sidelobes without reducing the image resolution or altering the phase information, is another example of a preprocessing algorithm reported to improve the detector performance in [37].

2.1.3 Prescreening/detecting

Prescreening algorithms search the whole image for potential target pixels. These algorithms often have adjustable parameters which control a tradeoff between missed detections and false alarms and are set to maximize the probability of detection, but present a high probability of false alarm. In addition, they must be fast to be applied in real time or near-real time systems. For these reasons, a further step of discrimination is needed [7].

There are several prescreening algorithms, and their choice depends much on the type of SAR image that is being processed. The resolution, complexity, intensity, amplitude and the number of looks are some important factors to take into account when choosing a prescreening algorithm. Most ship detection algorithms are based on statistical modelling of the background and then finding individual pixels or small groups of neighbouring pixels whose brightness values are statistically unusual.

One of the simplest prescreening algorithms for single channel SAR images is the global threshold algorithm, that searches for bright pixels in an image and declares pixels with values above a defined threshold as targets of interest, as used in [38, 39]. The brightness of the pixels is dependent on the target radar cross section (RCS), which, by its turn, is dependent on many factors including the target's material, the viewing aspect angle and the radar resolution. In low resolution SAR imagery, the target may only occupy part of a pixel, hence having a RCS lower than expected. Therefore, when building a detector, it is more common to opt for an adaptive threshold [7].

Adaptive threshold algorithms are among the most commonly used algorithms for target detection in SAR images. A key difference relatively to global threshold algorithms, is that the adaptive threshold algorithms look for bright pixels in the image, comparing them to their surroundings instead of the global picture. The threshold is defined according to the statistics of the surrounding area of the pixel being evaluated, that gets classified as a sample from a potential target if its value is above the threshold. The most common adaptive threshold detectors for SAR images are the CFAR detectors, and will be explained in a later section.

We will explore CFAR-based methods more thoroughly, since they are by far the most popular methods. Nevertheless, some context on non-CFAR methods will be provided.

CFAR-based methods

CFAR detectors are designed in a way that ensures a constant probability of false alarm, by selecting a threshold so that the percentage of background pixels with values above it is constant, resulting in a likewise constant false alarm rate (number of false alarms per unit area of imagery).

The various CFAR methods can vary on the type of the sliding window (usually referred to as stencil), that can be of fixed-sized or adaptive. Regarding the implementation technique, the various possible strategies include cell-averaging CFAR (CA-CFAR), smallest of CA-CFAR (SOCA-CFAR), greatest of CA-CFAR (GOCA-CFAR), and order statistics CFAR (OS-CFAR), among others. They can be further divided into two subclasses: parametric and nonparametric, depending on the method used to estimate the threshold (for the desired PFA) in the boundary ring or the approach taken to estimate the target signature (for the desired probability of detection (PD)). Two approaches are possible for the parametric subclass: background modelling only or background and target modelling. The non-parametric approach, on the other hand, does not assume any form for the background and target models. This designations are based on the taxonomy suggested by El-Darymli et al. [40].

Ideally, a CFAR detector should utilize a Bayesian approach, which for a zero-one cost reduces to the maximum *a posteriori* (MAP) criterion as

$$\Lambda_{MAP}(x) = \frac{P(\omega_T|x)}{P(\omega_B|x)} \lessgtr^{\omega_B}_{\omega_T} 1\,, \tag{2.1}$$

where ω_T is the target class, ω_B the background or clutter class, and $P(\omega_T|x)$ and $P(\omega_B|x)$ the posteriors of the target class and background class, respectively [40].

This is a binary classification problem, with x being a feature vector, containing the pixel values of the boundary ring of the sliding window, with guard cells centered in the region of interest (ROI). An example of a 9x9 sliding window, with a boundary ring represented in green, is depicted in Figure 2.1[40]. One should notice here that the center-most region in black is the target window size, with the pixels under test (PUTs), which, in the figure, is simply one pixel.

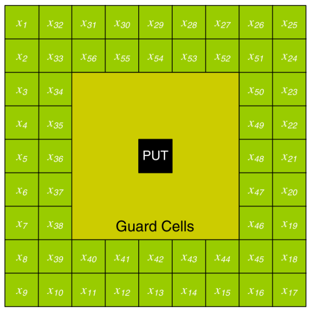


Figure 2.1: CFAR sliding window. From [40].

The target window size should be about the size of the smallest object that is expected to be detected, the guard ring window size should be about the size of the largest object expected to be detected, and the boundary ring window size should be sufficiently large to estimate the local clutter statistics accurately [7, 40], although choosing the dimensions of the stencil based only on the prior knowledge of target size yields a detection loss, due to the fact that the backscatter of the target is weakly linked with the target's geometry [41].

Parametric CFAR

As stated before, parametric CFAR models perform the detection threshold calculation based on the statistical modelling of either the background only, or based on the target and background statistical modelling altogether.

In the former approach, the CFAR algorithm performs one-class classification (anomaly detection) by assigning PUTs to the background if it finds that they are consistent with the modelled background distribution, otherwise the PUTs are labeled as detected. The threshold scaling factor α is adaptively computed based on the boundary ring of the sliding window, in order to attain a desired PFA. This computation is based on the likelihood or probability density function (PDF) of the background class and is given by

$$PFA = \int_{\alpha}^{\infty} p(x|\omega_B)\, dx\,, \tag{2.2}$$

where $p(x|\omega_B)$ is the chosen likelihood of the background class. The threshold is obtained by solving 2.2 for α.

In the latter approach, α is computed in order to achieve a desired probability of detection, resorting to the target likelihood $p(x|\omega_T)$

$$PD = \int_{\alpha}^{\infty} p(x|\omega_T)\, dx\,. \tag{2.3}$$

When a parametric CFAR algorithm uses an exponential distribution or a Rayleigh distribution to model the background clutter, it is referred to as a one-parameter CFAR, since it is characterized by only one parameter: the mean [40]. On the other hand, methods that utilize more complex distributions, characterized by two parameters, such as mean and variance or scale and shape parameters, are referred to as two-parameter CFAR. Examples of such distributions used in CFAR methods are the Weibull distribution [42], K-distribution [43], alpha-stable distribution [44, 45], and beta-prime ($\beta^{'}$) distribution [46], among others.

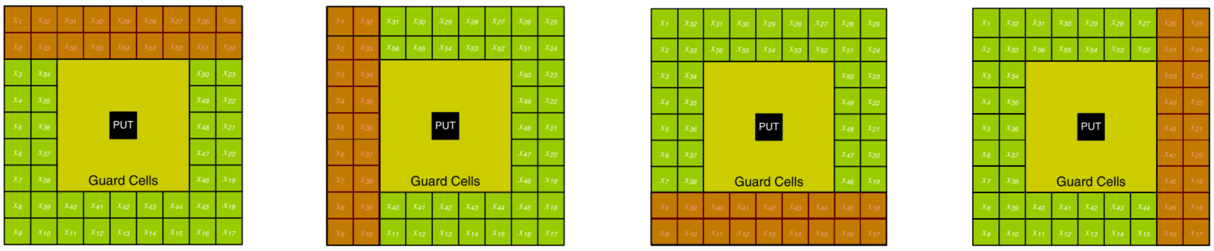
Most strategies applied in one-parameter CFAR algorithms can be applied in two-parameter CFAR. One of the possible strategies is the cell-averaging CFAR, or simply CA-CFAR. It was the first CFAR test, proposed in 1968 by Finn [47]. The adaptive threshold is the product of two components: Z and α, such as

$$Threshold = \alpha Z\,, \tag{2.4}$$

where Z is estimated from the boundary ring. The processed SAR image is complexed valued, with the form $I + jQ$. The CA-CFAR detector can be performed on the magnitude SAR image (i.e., $A =$

$\sqrt{I^2+Q^2}$) known as envelope detector or linear detector, on the power image (i.e., $P = A^2$), known as square-law detector or on the log-domain image (i.e., $L = 10\,log\,A^2$), known as a log detector. The choice of a Rayleigh distribution or an exponential distribution to model the clutter depends on whether the image is magnitude or power, respectively. The details for the estimation $\hat{Z}$ of Z will be omitted, since it is not considered relevant for this literature review. We simply state that it is obtained through the maximum likelihood estimate (MLE) of the arithmetic mean of the pixels in the boundary ring in the CFAR sliding window. The CA-CFAR then compares this mean with the pixels under test. The detection decision is contingent upon the threshold scaling factor α [40]. In CA-CFAR the targets are assumed to be isolated by at least the size of the sliding window, so that the number of targets in a stencil is, at most, one. Moreover, it assumes that the pixels in the boundary ring are independent and identically distributed and have a probability density function similar to that of the pixels under test. Since these assumptions do not hold in many real world scenarios, the performance of the method can suffer a great CFAR loss under circumstances that differ from the design assumptions [40]. This loss can be viewed as the required increase of in the signal to noise ratio (SNR) in order to maintain the desired PD [40].

There are variations of the CA-CFAR algorithm, such as the SOCA-CFAR and the GOCA-CFAR, which split the boundary ring of the sliding window into four leading and lagging windows. This separation is depicted in Figure 2.2.



(a) (b) (c) (d)

Figure 2.2: Leading (a, b) and lagging (c, d) windows. From [40].

In SOCA-CFAR, the smallest of the four means obtained with each of the leading and lagging windows is chosen for comparison with the PUTs. In GOCA-CFAR, the greatest of the means is chosen. SOCA-CFAR is designed to handle strong clutter returns in the boundary ring, although being prone to clutter edges. GOCA-CFAR outperforms CA-CFAR and SOCA-CFAR at clutter edges, but has weakened performance when in presence of strong returns in the boundary ring. Both of these two variations suffer from a greater CFAR loss, when compared to CA-CFAR, due to the fact that only part of the boundary ring is used.

To counter the issue of multiple targets in a stencil, the order-statistics CFAR was proposed [48]. OS-CFAR orders the pixels in the boundary ring according to their values, in an ascending order. Additionally, the Q'th percentile is chosen, instead of the average estimate used in CA-CFAR. The scaling factor α is estimated based on the clutter statistics of the boundary ring, similarly to previously mentioned CFAR methods. The choice of the value Q depends on the SAR data that is being analysed. For instance, in the original work [48], the value of Q that gives the best performance is $Q = 3/4$, while in [49] the best results are achieved with a value of $Q = 4/5$. The OS-CFAR outperforms CA-CFAR in heterogeneous clutter backgrounds and for contiguous targets [50, 51]. However, the performance of OS-CFAR degrades during clutter transitions. Switched order statistics CFAR (SWOS-CFAR), a variant of OS-CFAR, was designed to handle detection in non-homogeneous clutter and multiple interfering target scenarios [52]. It is able to determine whether the cells in the boundary ring belong to homogeneous or nonhomogeneous clutter and adaptively adjust the detection threshold for a desired PFA, being able to outperform standard OS-CFAR [53].

Nonparametric CFAR

Nonparametric CFAR models are more flexible and can better fit the real data faced by the detectors, since they do not assume any prior model for the background or the data [40, 54]. They use nonparametric methods to infer the model from the data. In [54], kernel density estimation (KDE) is used to infer the target and background models for CFAR detection, making use of the Parzen window kernel method to estimate the underlying PDF of the SAR data. In [55], nonparametric CFAR is implemented also using KDE to estimate maritime clutter distribution, using a target enhancement filter. Cui et al. [56] developed an asymptotic optimal bandwidth estimation method, that adaptively estimates the bandwidth with manually selected sub-regions. The method can be affected by wrongly selected training sub-regions and requires a complex procedure to obtain the threshold by iteratively calculating the PFA at each predefined pixel value level [55]. Leng et al. [9] proposed the bilateral CFAR algorithm, that performs detection based on the combination of the intensity distribution with the spatial distribution for the kernel density estimation, reducing the influence of sea clutter.

Non-CFAR methods

In this section, non-CFAR works for ship detection in SAR images are referenced. In the work of Ouchi et al. [57], the detection is based on a coherence image produced from the multilook SAR image via cross correlation between two SAR images extracted by two small-sized moving windows over the original image. In [58], the detection is based on genetic programming. Marino and Hajnsek [59] implemented a ship detection technique applied to fully polarimetric SAR data, based on the Geometrical Perturbation-Polarimetric Notch Filter [60].

An inshore ship detection method based on saliency and context information was proposed in [61]. This novel method aimed at filling the difficulty of detecting ships in harbors, presented by the existing algorithms. The method performs detection by generating superpixel region, followed by salient region detection. To decrease the false alarm rate, a discrimination framework based on the target size and the context information is applied.

A detection method using generalized likelihood ratio test (GLRT) in a K-distributed clutter is proposed in [62]. Others methods involving the GLRT have been proposed. Iervolino et al. [63] proposed a novel ship detection technique, composed of three steps: land mask rejection, detection and discrimination. Detection is performed using the GLRT, whereas discrimination is performed by rejecting some targets by removing the azimuth ambiguities and by gathering the target pixels in clusters. Iervolino and

Guida [8] implemented a ship detector using the GLRT, that takes into account both the sea clutter and the signal backscattered from the ship, in order to improve the performance of the detector, showing improvements when compared to CFAR algorithms, although computationally slower.

The Intensity Dual-Polarization Ratio Anomaly Detector (iDPolRAD) proposed by Marino and Iervolino [11] - initially developed for iceberg detection [10] - focuses on the improvement of the detectability of small vessels, basing its rationale on the fact that small ships possess a stronger cross polarization and a higher cross-over co-polarization ratio when compared to the sea.

2.2 Target detection with machine learning

In this section, a literature review on ship detection and discrimination with deep learning based methods is presented. The mentioned works utilize the current most advanced techniques used in the realm of computer vision, applied to the task of target detection in SAR images.

2.2.1 Deep learning

Deep learning for SAR automatic target recognition was first introduced by Chen and Wang [64], tested on the standard Moving and Stationary Target Acquisition and Recognition (MSTAR) ATR dataset [65]. In this work, authors found that the data lacked sufficient samples to train the network, so they implemented data augmentation techniques to reduce overfitting. This revealed to be a common problem in subsequent works by other authors [66–69].

CNNs applied to the detection and discrimination of vessels also started to emerge. Bentes et al. [70] applied a convolutional neural network (CNN) for ship-iceberg discrimination, tested on TerraSAR-X StripMap images, comparing the results with the performance of support vector machines, outperforming the latter with an average f1-score of 97%. The CNN was composed of two convolutional layers followed by pooling layers and a fully connected layer. A custom nonlinear normalization function was also used in order to reduce isolated, strong backscattering signals from targets. Schwegmann et al. [71] applied highway networks [72] (a specific type of deep neural networks) for ship discrimination in a RADARSAT-2 and Sentinel-1 dataset, composed of 21x21 images with ships, with ship-like targets (false positives) and ocean areas. The goal of using this type of network is to attenuate the problem of vanishing gradients for very deep networks, by allowing the transfer of information across multiple layers. It achieved an accuracy of 96.67%. In [73], a CNN with six convolutional layers, three pooling layers and two fully connected layers is applied to potential targets detected by the Single Shot Multi-box Detector (SSD). Bentes et al. [74] built a dataset consisting of ships, platforms and harbors, applying to it a CNN model with four convolutional layers, four pooling layers and a fully connected layer, achieving an accuracy of 94%, a performance far superior compared to traditional machine learning models.

Several popular top performing detection methods are based on the region-based convolutional neural network (R-CNN) [75], as well as on the fast R-CNN [76] and faster R-CNN [77].

R-CNNs use selective search to extract region proposals, proceeded by a recognition technique such as a CNN to classify them, being characterized by a tedious and time-consuming process, as well as slow training [14]. Fast R-CNNs reduce the computational complexity and improve the performance of R-CNNs by directly using the softmax function instead of support vector machines. Moreover, region of interest polling also improves the performance of R-CNNs [14]. Although presenting good results, they have limited speed performance due to bottlenecks in the proposed areas [14]. Faster-RCNNs unify the candidate area generation, feature extraction, classification and location refinement into a deep network framework, and implement a complete end-to-end target detection model [14]. It consists of three main networks: a convolutional network to extract feature maps, the region proposal network (RPN), and a network using these proposals for object classification and bounding box regression.

Kang et al. [78] implemented a faster R-CNN and fused the deep semantic and shallow high-resolution features in both region proposal network and region of interest layers, improving detection for small-sized ships. In [16], a faster R-CNN based framework was used to carry out the detection task in conjunction with a CFAR method to detect small targets, reevaluating bounding boxes with relatively low classification scores in the detection network. In [79], a densely connected multiscale neural network based on the faster R-CNN framework was proposed to solve multiscale and multiscene SAR ship detection. Instead of using a single feature map to generate proposals, they densely connected one feature map to every other feature maps from top to down to generate proposals from each fused feature map. They also used a training strategy to reduce the weights of the easiest examples in the loss function, so that the harder ones were given more attention in training, to reduce false alarms. Li et al. [80] proposed multiple strategies in order to improve the results of the faster R-CNN applied to ship detection, such as feature fusion, transfer learning and hard negative mining. They also provided a new dataset called SAR ship detection dataset (SSDD). Lin et al. [81] propose a new R-CNN based approach to improve ship detection, by using a squeeze and excitation mechanism [82]. First, the feature maps are extracted and concatenated to obtain multiscale feature maps with a VGG network [83], pretrained with the ImageNet dataset. After RoI pooling, an encoding scale vector is generated from subfeature maps. This vector is ranked and only the top values are preserved, setting the others to zero. The subfeature maps are then calibrated using the scale vector, suppressing the redundant subfeature maps, improving the detector performance. An improvement of 9.7% on the F-1 score compared to the state-of-the-art method is claimed, along with an increase of 14% in speed.

The You Only Look Once (YOLO) method proposed by Redmon et al. [84], is a new approach for object detection that uses a single neural network to predict bounding boxes and class probabilities directly from full images in one evaluation. The method frames object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. YOLO presented a remarkable speed and the reduced likeliness of predicting false positives on background. Moreover, since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance. An improved version of the YOLO, YOLOv2, was proposed by Redmon and Farhadi [85], using a novel, multi-scale training method. This version could run at varying sizes, increased speed and allowed to easily tune the tradeoff between speed and accuracy, outperforming the state-of-the-art methods at the time, such as the faster R-CNN, ResNet [86] and the Single-Shot-Detector (SSD).

YOLOv2 was first used for SAR ship detection in [13]. The framework was trained on three Sentinel-1 datasets, two with images with ships and one with single ship images. They implemented transfer learning by training the network on the third dataset and then dropping the last layer in order to further train on the first two datasets. Chang et al. [14] implemented a ship detector using a YOLOv2 framework, as a base to implement vessel detection and adjust the parameters to achieve high accuracy in near real-time, on the SSDD dataset. They also propose a modified YOLOv2 network, the YOLOv2-reduced, with less layers. The YOLOv2 method achieved a detection speed 5.8 times faster than the faster R-CNN. The YOLOv2-reduced goal was to be more suitable for real-time detection, performing at a 2.5 times faster speed than the YOLOv2, whilst maintaing a decent detection accuracy.

Redmon and Farhadi [87] proposed an improvement on YOLOv2, the YOLOv3. It is composed of a bigger network and more accurate than the previous one, achieving similar accuracy results compared to SSD, but three times faster. However, to the extent of our knowledge, works using YOLOv3 applied to SAR ship detection have not yet been published.

The SSD is another CNN-based method widely applied in detection in SAR images. This method, proposed by Liu et al. [88], aims to discretize the output space of bounding boxes into a set of default boxes, over different aspect ratios and scales per feature map locations. At prediction time, the model ranks each default box for the presence of each object category, and adjusts the box to better match the object shape. In the original work, it is claimed to outperform comparable faster R-CNN models. Wang et al. [89] propose an SDD based method to simultaneously detect ships and their orientation, by using rotatable bounding boxes, adding an attention module to the model's six prediction layers. This module takes into account the channel and space information of ships, helping in their detection. In addition, angular regression is used to predict angles without increasing the computational load. In [90], two SSD models are implemented: the SSD-300 and SSD-512, with input sizes of 300x300 and 512x512, respectively. The main goal pointed out by the authors is the training of an effective model without much data, by using data augmentation and transfer learning. The model is built with a VGG16 network, pretrained on the PASCAL VOC dataset. In [73], a modified SSD with a multi-resolution input (MR-SSD) is developed. The whole workflow includes several stages, such as sea-land segmentation, cropping with overlapping, detection with the MR-SSD model, coordinates mapping and consolidation of predicted boxes. The authors also highlight the ability of the model to classify several types of marine targets, such as boats, cages, cargos, containers, towers, platforms, tankers and windmills.

Residual networks (ResNets), proposed by He et al. [86], were a major breakthrough for deep learning based computer vision algorithms. ResNets use residual connections, that consist of reinjecting previous representations into the downstream flow of data by adding a past layer output to a later output layer, which helps prevent information loss along the data-processing flow. These connections allow increasing the accuracy of the network by increasing its depth.

In [91], a ship detection method based on an attention mechanism is proposed, with a backbone network built with Inception-ResNet [92] modules to obtain multilevel target mapping features. The saliency of the mapping features is enhanced with an attention mechanism to obtain saliency feature maps. Then, these features expressed at different depths are fused. On the fused feature maps, the locations and confidence scores of the targets are predicted. The final step is to filter the predicted boxes via Soft-Non-maximum Suppression (Soft-NMS) [93]. The main aspects of the built end-to-end framework are the increased accuracy in detecting densely arranged ships, its capability to address multiscale characteristics of ships and feasible real-time detection speed.

The RetinaNet [94] is a network architecture that consist of three components: a backbone network and two subnetworks (one for classification and another for box regression). The backbone computes a convolutional feature map over an entire input image and can be a commonly used CNN such as ResNet, VGG, Inception [95] or DenseNet [96], followed by a Feature Pyramid Network (FPN). The FPN is able to construct a multi-scale feature pyramid to be used for detection of objects at different scales [94]. Wang et al. [97] built a ship detector using RetinaNet with VGG and ResNet for the tested backbone networks. The FPN is used to extract multi-scale features for both ship classification and location. Focal loss is used to address the class imbalance present in the data and to increase the impact of the harder examples during training, obtaining good detection results, with over 96% of mean average precision.

2.3 Representation learning

In this section, we provide an overview of the literature related to the application of autoencoders to the analysis of SAR images. As it will become apparent later, autoencoders have been playing an increasingly important role in the feature extraction process of deep learning based frameworks for classification in SAR imagery.

2.3.1 Autoencoders

Autoencoders (AEs) are neural networks trained in an unsupervised manner to reconstruct their inputs at the outputs [98]. They are composed of two parts: an encoder and a decoder. The encoder maps input data $x \in \mathbb{R}^d_x$ to a latent space representation $z \in \mathbb{R}^d_z$ and the decoder maps back from the latent code z to input space. Tipically the latent code z has a lower dimensionality than the input space x which forces the autoencoder to learn a compressed representation of the input data. Learning is conducted by minimizing a loss function $L(x, \hat{x})$ (such as the mean squared error), that measures the dissimilarity between the inputs x and outputs $\hat{x}$.

Kang et al. [99] proposed a feature fusion algorithm for SAR target recognition based on a stacked autoencoder (SAE) [100]. SAE is a type of unsupervised learning network that converts raw data into more abstract expressions through a non-linear model and fuses features by optimization algorithms [99]. The framework implemented in [99] firstly extracts 23 baseline features and Three-Patch Local Binary Pattern (TPLBP) [101] features, that describe the local and global aspects of the image with less redundancy, providing richer information for feature fusion. The goal is to have distinctive low-dimensional features to provide complementary information for the SAE. Those features are cascaded and fed into an SAE, which is pre-trained by greedy layer-wise training and fine-tuned with a softmax classifier. The model is applied to the classification of targets in the MSTAR dataset.

In [102], a deep convolutional AE (DCAE) is used to extract features and perform classification automatically. DCAE is a type of AE that uses CNNs in its architecture. The DCAE in [102] consists of eight layers. It has a handcrafted first convolution layer, such as gray-level co-occurrence matrix and Gabor filters, and a handcrafted second layer of scale transformation that integrates correlated neighbor pixels. The four following layers are based on SAEs to optimize features and classify, and the final two layers are for postprocessing. The method was applied to terrain surface classification on TerraSAR-X images, with the objective to make up for the absence of effective feature representation and the presence of speckle noise in SAR images. Geng et al. [103] proposed a similar framework, with a deep and contractive NN for SAR image classification. This method additionally includes the histogram of oriented gradient descriptors as handcrafted kernels. The AE is trained with a penalty that important information between features and labels, as well as a contractive restriction that enhances local invariance. In this work, the authors found that speckle reduction yielded the worst results, suspecting that the smoothing of the speckle might have hidden some relevant information.

In [104], a SAE combined with superpixels is proposed for terrain surface classification in PolSAR images. In this line of work, multiple AE layers are trained on a pixel-by-pixel basis, and superpixels are constructed based on Pauli-decomposed pseudocolor images. The outputs of the SAE are used for k-nearest neighbor (KNN) clustering of the superpixels. In [105], stacked sparse AEs (SSAEs) are applied to PolSAR data for classification of terrain surface, aimed to learn deep spatial sparse features automatically.

Li et al. [106] propose the usage of a CAE as an unsupervised learning method to extract high-level features, in conjunction with a shallow neural network that works as a classifier, presented as a fast training method with little loss of recognition rate, tested on the MSTAR dataset.

In [107], denoising AEs are used for unsupervised feature learning, learning local and high-level representations from the local neighborhood of the pixels. The proposed framework is used to detect changes on multi-spatial-resolution images. Gong et al. [108] built a feature learning and change feature classification for ternary change detection in SAR images. The method utilizes a sparse AE to transform log-ratio difference image into a feature space that allows the extraction of key changes and the suppression of outliers and noise. The learned features are clustered into three classes to be passed to a CNN classifier to be used as pseudo-labels for training. The CNN works as a change feature classifier.

Deng et al. [109] developed a SAR ATR system with autoencoders, adding a supervision constraint based on Euclidean distance in order to get the most out of a limited training set.

In [110], a novel approach using AEs and convolutional AEs to perform representation learning is proposed. The framework, named Wishart-AE and Wishart-CAE, introduces novelty by using the coherence matrix of POLSAR data as the input of the network and by applying the Wishart distance [111] to measure the similarity between the inputs and outputs of the model.

Geng et al. [112] propose an end-to-end classification framework, named deep recurrent encoding neural networks (DRENNs). They use a long-short term memory (LSTM) architecture to extract contextual dependencies of the image and learn contextual dependencies, using 2-D patches of the image transformed into 1-D sequences. After the LSTM, non-negative and Fisher constrained autoencoders

(NFCAEs) are proposed to improve feature discrimination and perform classification.

In [113], an unsupervised technique using variational autoencoders is used to detect avalanches in Sentinel-1 SAR images. The VAE, proposed by Kingma and Welling [114], is a deep generative model composed of a stochastic encoder and decoder, that models the relationship between the inputs and outputs. In [113], a VAE is trained only on images without avalanches, to learn feature representation of normal data, and treat avalanches as anomalies, that are very rare.

Xu et al. [115] propose the usage of VAEs, constructed with residual connections, to extract useful features, and using the various dimensions of the obtained feature latent space, to feed into supervised classifiers, such as SVMs and KNNs.

Chapter 3

Materials and methods

In this chapter, we start by introducing the theoretical and mathematical frameworks that compose the foundations of the methods used to achieve the goal of this thesis. Following this background analysis, the delineated approaches to solve the problem at hand are briefly presented.

3.1 Theoretical background

In this section, we delve into the mathematical foundations of the several methods and algorithms used throughout this work. First and foremost, convolutional neural networks are explained thoroughly, since they are one of the main building blocks of the proposed framework. Next, the theoretical underpinnings of autoencoders are explained, followed by a brief introduction to principal component analysis (PCA) and the used clustering methods: K-means and Gaussian Mixture Models (GMMs).

3.1.1 Neural networks

Deep feedforward neural networks

Deep feedforward networks are the basis of deep learning models. Such networks have the objective to learn an approximation of some function f^*. For instance, for a classifier that assigns a category to some input data x, such that $y = f^*(x)$, the neural network tries to learn the best function approximation $y = f(x;\theta)$, mapping the inputs x along with some learned parameters θ. The convolutional neural networks - which will be discussed in detail later in this chapter - are a special kind of feedforward neural network, used in many different computer vision tasks, such as object recognition [116]. Feedforward neural networks are typically represented by wiring together many different functions. A neural network might have, for example, three functions f_1, f_2 and f_3, connected in a chain, to form $f(x) = f_3(f_2(f_1(x)))$. This kind of structure is the most commonly used in neural networks. In this example, f_1 is called the first layer of the network, whereas f_2 is the second layer, and so on. The more layers the bigger the network's depth, hence the name deep learning for the process of learning with these structures.

The final layer of a neural net is called the output layer. At this level, it is usually ”shown” to the network what the values of its outputs should be, given some input, through training examples that usually come in (x, y) pair values, where x are the inputs and y are the outputs, or the answers for the input data. Training examples ”tell” the network what output values its final layer should produce. However, they don’t specify what values the remaining layers must produce. Instead, during the process of training, the learning algorithm must tweak the parameters of the other layers in order to use them to achieve an output close to the examples that are shown to it, iteratively refining its approximation of f^*. These layers are called the hidden layers. Each hidden layer is usually vector-valued and their dimensionality determines the width of the network.

In a neural network composed of fully connected (FC) layers, also known as dense layers, the output of each hidden layer is computed by applying a nonlinear function to the outputs of the previous layer, multiplied by a set of weights W with an added set of biases b, such that

$$a^{(l)} = g(W^{(l)^T} a^{(l-1)} + b^{(l)}) \, , \tag{3.1}$$

where $a^{(l)}$ is the output value at the layer l, $W^{(l)}$ and $b^{(l)}$ are the set of weights and the set of biases for the layer l, respectively, and g is a nonlinear function, commonly referred to as the activation function. There are several options for the nonlinearity g and deciding what is the best one is an active field of research, but the default recommendation in modern neural networks is the rectified linear unit (ReLU) function [117, 118], defined by the function $g(z) = max\{0, z\}$. For the output layer, the activation function usually varies in accordance to the problem at hand. $a^{(0)}$ is simply the input data x.

In Figure 3.1, the general architecture of a dense deep feedforward neural network is presented, for a network with d layers, each with l_i hidden units, with $i = 1, \ldots, d$, and input size k.

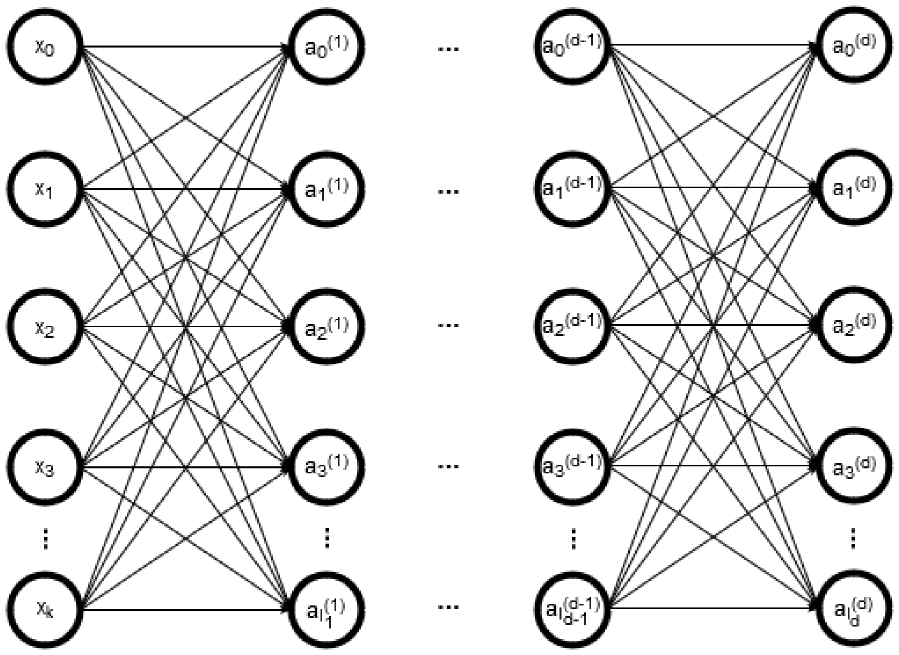


Figure 3.1: MLP general architecture.

Convolutional neural networks

Convolutional neural networks, also known as CNNs or ConvNets, are a class of neural nets, specialized to process data that has a known, grid-like topology, that have proven to be very efficient in solving computer vision tasks, such as image recognition and classification [12]. The name of these networks comes from the mathematical convolution operation that is performed instead of general matrix multiplication, in at least one of the layers of the network. Technically, this operation does not correspond precisely to the definition of convolution used in other fields such as pure mathematics, corresponding instead to the cross-correlation function, which is very similar to the classical convolution function but without kernel flipping. Since it is common among machine learning practitioners to simply call this operation convolution, we will do it so throughout the rest of this work. For two-dimensional data I and a two-dimensional kernel K, the convolution operation is given by

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(M,n)\,. \tag{3.2}$$

This operation is illustrated in Figure 3.2.

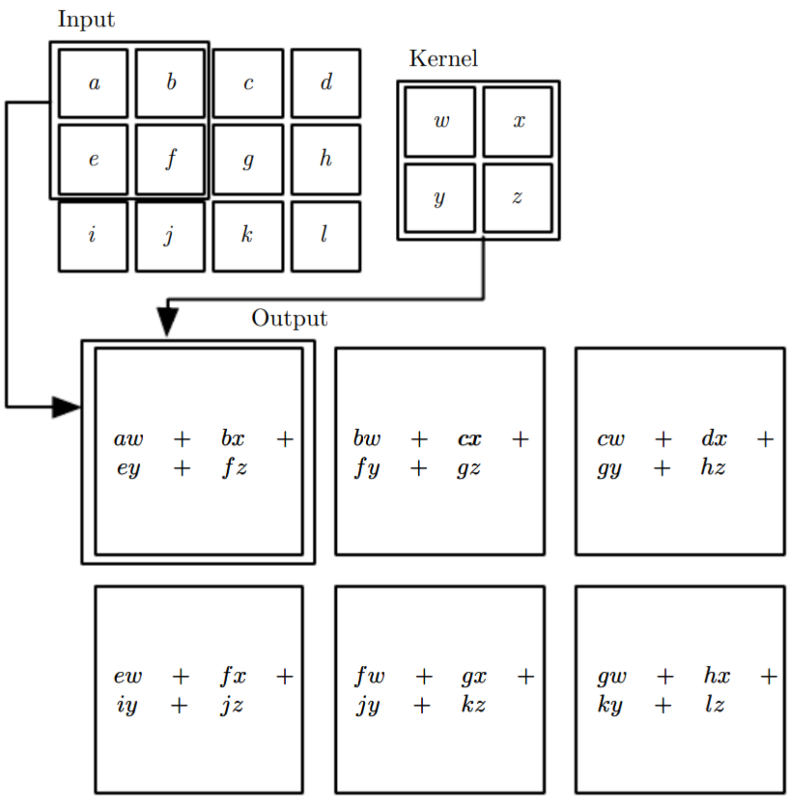


Figure 3.2: Convolution without kernel flipping. The arrows point to the output formed by applying the kernel to the upper-left element of the input. In this example, the outputs correspond exclusively to positions where the kernel lies entirely within the image, often called ”valid” convolution. From [116].

The input data, however, has often more than two dimensions. One example are colored images, which have three color values for each pixel, for red, green and blue channels. Each convolutional layer accepts an input tensor with dimensions $W_1 \times H_1 \times D_1$ (width, height and depth) and has four hyperparameters: the number of kernels (also referred to as filters or feature detectors) K, their size F (assuming square filters), the stride S, which is the number of pixels by which the filter is slided, and the amount of padding P. Padding consists in adding a frame of pixels around an image, usually zero-valued. The filters' depth is equal to the depth of the input data. The term hyperparameter is used in machine learning to refer to values that are chosen or set before training.

After passing through such convolutional layer, the dimensions of the output data, $W_2 \times H_2 \times D_2$, are given by

$$W_2 = \frac{W_1 - F + 2P}{S} + 1,\ H_2 = \frac{H_1 - F + 2P}{S} + 1,\ D_2 = K. \tag{3.3}$$

Convolutional layers are useful because they have the ability to learn local patterns in the images and recognize them anywhere, by learning linear transformations.

After the convolution operation, the produced set of linear activations are run through a nonlinear activation function, such as the ReLU. The next stage is to use a pooling layer to further modify the output data. Spatial pooling aims at reducing the dimensionality of each feature map, while retaining the most relevant information, by replacing the values of the network at a certain point with a summary statistic of the nearby outputs. Pooling helps in making the representation approximately invariant to small displacements of the input, meaning that small translations of the input do not translate in great variations of the pooled outputs [116]. By progressively reducing the spatial size of the data, it helps reducing the number of parameters and the amount of computation in the network. For example, max pooling takes the maximum value of a rectangular neighborhood of pixels, and is one popular pooling method. The most common is a max pooling layer with filters of size 2x2 with a stride of 2, downsampling every depth slice by 2 along both width and height, discarding 75% of the activations. This operation is depicted in Figure 3.3.

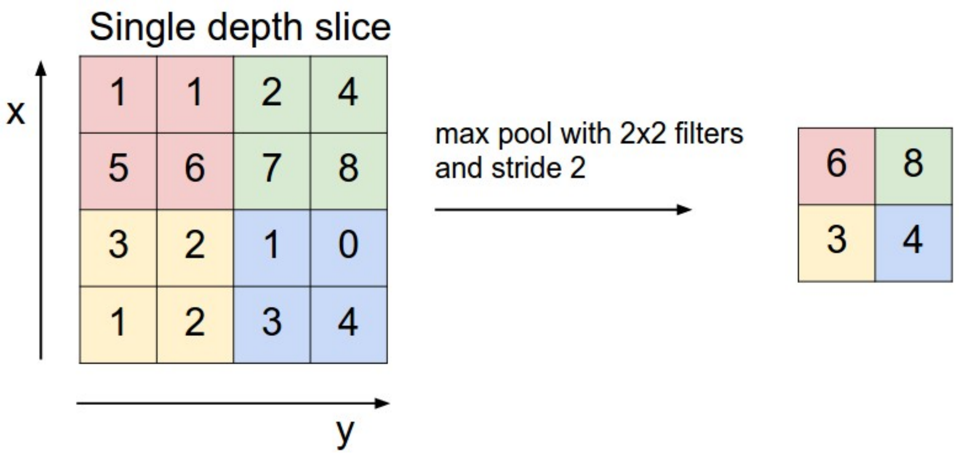


Figure 3.3: Max pooling operation with a 2x2 filter and a stride of 2. From [119].

The last layer of a CNN is usually an FC layer, with an activation function chosen according to the goal that the network is trying to achieve.

Training a neural network

Having chosen the architecture of the network, it is then necessary to learn the parameters that give the best results for a given dataset. This process of learning parameters that best fit the given data is called training.

The available dataset is usually split into three sets: the training set, which usually contains most part of the available data and is used to train the network; the validation set, that is used to evaluate the performance of the model on unseen data during training and choose the set of hyperparameters that lead to the best results; and the test set, which is usually used after the model is trained, to evaluate its unbiased performance on unseen data. Although the model does not update its parameters directly based on the validation set during training, since the final choice is based on the results on this data, it would not be right to judge its generalization capability based on them. Consequently, a final step of testing the results on the test set is performed.

During training, for each input example (x, y) in the training set, the network performs the forward computations (forward pass), producing an output prediction $\hat{y}$. A chosen loss function is used to compute the difference between the ground truth y and the prediction $\hat{y}$. The network parameters are updated based on this value, in order to optimise the loss function, using an optimisation algorithm. Most training algorithms split the whole data into minibatches of data, containing only a few examples. Each time the training algorithm performs a complete pass through all the data in the dataset an epoch as passed, and it is usual to train a neural network with several epochs, meaning the algorithm ”sees” the training data many times.

Ideally, training would eventually lead to the set of parameters that correspond to the global minimum of the loss function. However, since neural networks generally produce non-convex functions, finding this global minimum is very difficult.

Optimisation

Neural networks are usually trained by using gradient-based optimisers, that iteratively drive the cost function to a low value. The general idea behind these algorithms consists in using the negative gradient of the loss function (that points in the direction in which the loss function decreases the fastest), with respect to the parameters, and then taking small steps in that direction, repeating the process until a local minimum is found. The cost function is often composed of a sum of per-example loss functions, for all examples in the training set, leading to a computationally expensive gradient calculation. To overcome this problem, optimisation algorithms, such as the stochastic gradient descent (SGD), typically use small sets of examples (minibatches) at a time to compute an estimate of the gradient, treating it as an expectation (as opposed to batch gradient descent, that uses all examples at once to compute the gradient). Each minibatch $B = \{x^{(1)}, \ldots, x^{(m')}\}$ is drawn uniformly from the training set.

The estimate of the gradient in SGD is formed as

$$g = \frac{1}{m'} \nabla_\theta \sum_{i=1}^{m'} L(x^{(i)}, y^{(i)}, \theta)\,, \tag{3.4}$$

where L is the loss function and θ the parameters of the network. The SGD algorithm then follows the gradient downhill:

$$\theta \leftarrow \theta - \epsilon\, g, \tag{3.5}$$

where ϵ is the learning rate, a positive scalar that determines the size of the update step.

The size of the minibatches influences the performance of the network, and its choice is usually driven by the following factors:

- Larger batches provide a more accurate estimation of the gradient.
- Very small batch sizes can lead to underutilisation of multicore systems.
- The amount of memory usage usually scales with batch size, usually being a limiting factor.
- It is common for graphics processing units (GPUs) to have better runtime using power of 2 batch sizes.
- Small batches can offer a regularizing effect [120], improving generalization ability.

It is also of extreme importance to choose the minibatches randomly, so that the gradient estimate of each batch is unbiased. For this, it is common practice to randomly shuffle the dataset before splitting it into batches.

Although the computation of an analytical expression for the gradient might be feasible, evaluating it numerically can be computationally expensive. The back-propagation algorithm [98], or backprop, does this calculation using a rather simple and inexpensive procedure, allowing the information to flow backwards from the end of the network, in order to compute its value. The gradient of the loss with respect to each input variable is computed applying the chain rule, computing the gradient one layer at a time and avoiding redundant calculations of intermediate terms.

The choice of the learning rate - often regarded more as an art than a science - is a crucial step in the training process of a neural network. In practice, it is common to schedule a learning rate decay over the course of training, defined as

$$\epsilon_k = (1 - \alpha)\epsilon_0 + \alpha\epsilon_\tau\,, \tag{3.6}$$

where ϵ_k is the learning rate at iteration k, ϵ_τ the learning rate at iteration τ, point from which the learning rate is kept constant, and $\alpha = \dfrac{k}{\tau}$. Setting ϵ_0 can be tricky. With a too small ϵ_0, learning can get stuck at a high loss value. If it is too high, on the other hand, the learning curve can show violent oscillations, possibly leading to an increase in the loss value instead of a decrease, and get driven away from the optimisation goal. Usually τ is set to a number of iterations that allows a few hundred passes through the training set, and ϵ_τ should be set to roughly 1% of the value of ϵ_0.

Since learning can sometimes be slow, the method of momentum [121] may be applied to accelerate learning, specially when the algorithm of choice is faced with high curvature, small but consistent gradients, or noisy gradients. The momentum algorithm takes the past values of the gradients and accumulates them on an exponentially decaying moving average, continuing to move in their direction. This way, the size of each update step depends on the direction and size of a previous sequence of gradients, being larger when many updates point in the same direction.

Parameter initialisation is also of key importance when training neural networks. The algorithms used to train neural networks are iterative, meaning that a starting point must be chosen by the user. This choice strongly influences the behavior of the algorithm, and it can determine whether the algorithm converges at all. All the weights in deep learning models are almost always initialised randomly, drawn from a Gaussian or uniform distribution. The scale of the initialised weights is also a major factor: larger weights will help avoiding redundant units, while avoiding loosing signal along the network. However, if weights are initialised with too large values, they may result in exploding values during forward propagation or backprop. The biases are typically set to constants based on some heuristic, usually initialised with the value of 0.

A standard optimisation technique applied in deep learning is batch normalisation, proposed by Ioffe and Szegedy [122]. The general idea is to force the activations throughout the network to take on a standard Gaussian distribution at the beginning of training, introducing both additive and multiplicative noise on the hidden units during training, improving both optimisation and regularisation. This technique is performed after the convolutional/dense layers, but before applying the activation functions, by applying the following:

$$\begin{aligned} H' &= \frac{H-\mu}{\sigma} \\ \mu &= \frac{1}{m}\sum_i H_i \\ \sigma &= \sqrt{\delta + \frac{1}{m}\sum_i (H-\mu)_i^2}\,, \end{aligned} \tag{3.7}$$

where H is the mini-batch of activations of the layer to be normalised, μ and σ vectors containing the mean and standard deviation of each unit, respectively, and δ a small positive value added to avoid having σ close to zero, avoiding numerical issues. The subtraction and division applied in the first equation of 3.7 are element-wise within each row of H, so $H_{i,j}$ is normalised by subtracting μ_j and dividing by σ_j

This technique allows the network to converge for a broader set of hyperparameters and makes it more robust to poor weight initialisation.

Algorithms with adaptive learning rates

The learning rate is arguably one of the hyperparameters that is the most difficult to set, due to its significant impact on the performance of the model. Naturally, algorithms with adaptive learning rates have emerged.

The AdaGrad algorithm, proposed by Duchi et al. [123], adapts the learning rates of all model parameters individually, dividing them by the square root of the sum of all their past squared values, meaning that parameters with large partial derivatives of the loss have their learning rates greatly decreased, whereas parameters with small partial derivatives have a small decrease in their learning rates. Although the AdaGrad should theoretically converge well in convex scenarios, the square gradient accumulation from the earlier stages of training can cause the learning rates to decrease too rapidly and excessively.

RMSProp [124] modifies AdaGrad by using an exponentially weighted moving average instead of gradient accumulation. This modification allows RMSProp to converge rapidly after finding a convex bowl, in a non-convex optimisation problem, whereas with AdaGrad, this convex bowl might never be found, since the learning rate can quickly become too small. RMSProp has been one of the go-to optimisation methods for training deep learning structures.

Adam [125], short for adaptive moments, is one of the most broadly used optimisation algorithms to train neural networks, that combines the fundamentals of RMSProp along with momentum, with some additional distinctions. In Adam, momentum is implemented with an estimate of the first order moment of the gradient, applied to the rescaled gradients. In addition, bias corrections to the first-order and second-order moments are included, to account for their initialization at the origin. This algorithm is usually robust to hyperparameter choices, usually working well with the recommended learning rate. The algorithm is described in Figure 3.4, with all operations on vectors being element-wise.

```
Require: $\alpha$: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters $\theta$
Require: $\theta_0$: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1$^{st}$ moment vector)
  $v_0 \leftarrow 0$ (Initialize 2$^{nd}$ moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while $\theta_t$ not converged do
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep $t$)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return $\theta_t$ (Resulting parameters)
```

Figure 3.4: Adam algorithm. From [125].

Although there is no consensus on what is the best optimisation algorithm to opt for, popular choices include SGD, RMSProp and Adam.

3.1.2 Regularisation

Regularisation consists in applying techniques to improve the generalisation capability of the model. It is a key task to make the model perform better on unseen data, even if it means to increase its training error. Regularisation techniques can put extra constraints on the model, such as restrictions for the parameter values, add extra terms to the cost function, encode some kind of prior knowledge or even express a generic preference for a simpler model. Simpler models usually promote better generalization, associated with a lower variance and a higher bias. Some methods can also combine multiple hypotheses that explain the training data.

Many regularisation approaches limit the model capacity by adding a parameter norm penalty $\Omega(\theta)$ to the cost function. The resulting regularised cost function is given by

$$\tilde{J}(\theta; X, y) = J(\theta; X, y) + \alpha\Omega(\theta)\,, \tag{3.8}$$

where α is a non-negative hyperparameter that defines the weight of Ω. Increasing α means increasing regularisation. When applying regularisation to neural networks, it is usual to use a parameter norm penalty that only penalises the weights, leaving the biases unregularised.

Next, some of the most common regularisation techniques used in neural network training are presented.

L^2 regularisation

L^2 regularisation, commonly known as weight decay, drags the weights w to the origin, by adding the regularising term $\Omega(\theta) = \frac{1}{2}\alpha w^T w$ to the objective function, directly penalising the squared magnitude of all parameters. The resulting cost function is then given by

$$\tilde{J}(w; X, y) = J(w; X, y) + \frac{\alpha}{2} w^T w\,. \tag{3.9}$$

This method heavily penalises high valued weights, giving preference to more diffused ones, encouraging the network to use all of its inputs instead of using only some of them.

L^1 regularisation

L^1 regularisation is another popular form of regularisation, that adds $\Omega(\theta) = \sum_i |w_i|$ to the cost function. It has the property of leading the values of some weights to values very close to zero during optimisation, making the regularised neurons of the network almost invariant to noisy inputs.

Max norm constraints

Enforcing an absolute upper bound on the magnitude of the weight vector for every neuron is another regularisation method. The constraint is enforced using projected gradient descent. Since the weight updates are always bounded, the network cannot explode even when using too high learning rates.

Dropout

Dropout, proposed by Srivastava et al. [126], is a simple yet very powerful technique to prevent deep neural networks from overfitting, that complements other regularisation methods such as the L^2 regularisation. During training, dropout is implemented by keeping each neuron active with some probability

p, a tunable hyperparameter. This way, each neuron has the probability p of staying active, being set to zero otherwise. At each training iteration, a subnetwork is then sampled and only the parameters of this subnetwork are updated based on the input data. This way, the network is impeded to heavily rely on specific neurons, and learning interdependence amongst them is reduced, increasing generalisation power. Dropout is depicted in Figure 3.5.

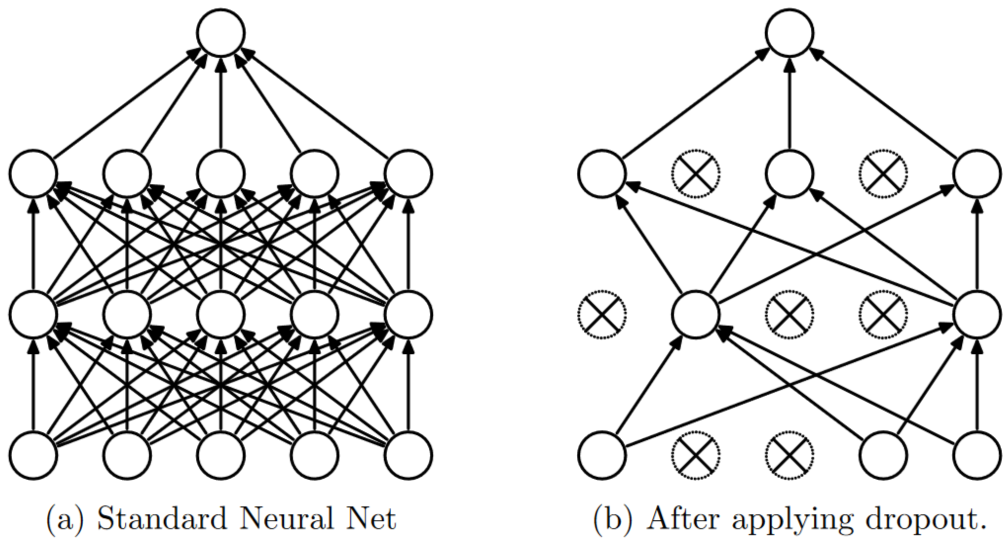


Figure 3.5: General idea behind dropout, taken from [126].

Batch normalisation

Although the primary purpose of batch normalisation is to improve optimisation, it also has a regularising effect due to the additive and multiplicative noise that it introduces.

Early stopping

When the model that is being training has sufficient representational capacity, it is common to see its training error steadily decreasing over each epoch, with the validation error starting to rise at some point in time. Since the objective is to have the model that has the best generalisation power, the ideal set of parameters is the one that produces the best validation error.

To keep these parameters, early stopping is used. It is a simple technique that keeps storing the model parameters that have given the best results for the validation set until the most recent training point. This way, When training has finished, the returned parameters are the ones for which the network has performed better.

3.1.3 Autoencoders

Autoencoders [98] are feedforward neural networks trained to reconstruct their inputs at the outputs, in an unsupervised manner. Unsupervised learning is the process of training a neural network without using the labels of the training set. An autoencoder network is usually composed of two parts: an encoder and a decoder. The encoder is responsible to map the input data $x \in \mathbb{R}_x^d$ to a hidden space representation $z \in \mathbb{R}_z^d$, through some encoding function, such that $z = f(x)$. The decoder part of the network then maps back from the hidden code z to input space, producing a reconstruction of x, $\hat{x} = g(z)$. The general architecture of an autoencoder is depicted in Figure 3.6.

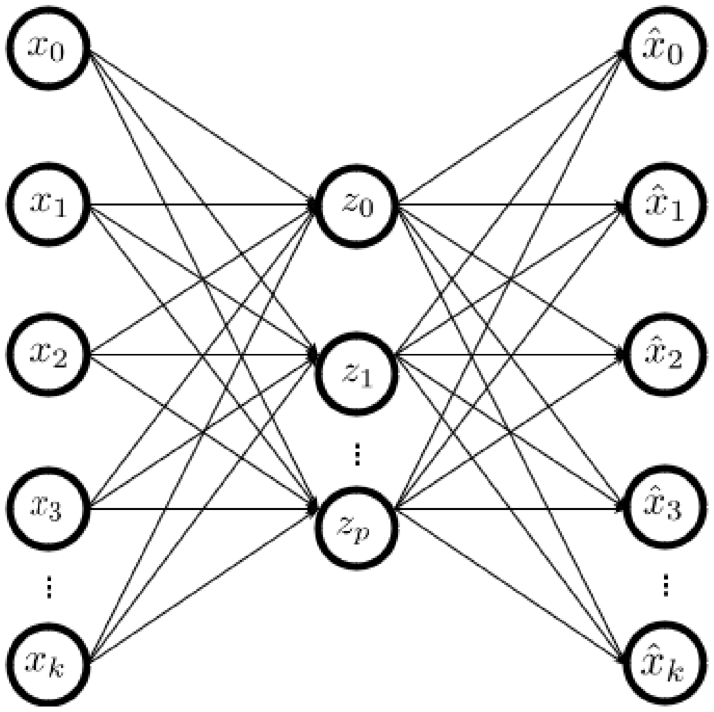


Figure 3.6: General architecture of an autoencoder.

A regulariser term can be added to ensure that the model does not overfit the training set, and effectively learns a useful representation of the data.

Typically the hidden code z has a lower dimensionality than the input space x, which forces the autoencoder to learn a compressed representation of the input data. When an autoencoder possesses a hidden code with dimensions smaller than the inputs, it is called undercomplete.

Using a compressed code with lower dimensions can make the autoencoder learn the most salient features of the data. However, if the encoder and decoder of the model are given too much capacity, they can learn how to map the inputs to the outputs, regardless of the dimensionality reduction observed in the network. The same goes to architectures with hidden code dimensions equal to or higher than the input dimensions, cases in which even a linear encoder and a linear decoder could learn to mimic the inputs without learning anything useful. These situations can be avoided by applying regularisation to the autoencoder. Instead of limiting the power of the model by capping the capacity of the encoder and decoder, a regularising term that encourages the network to learn other properties can be added to the loss function, leading the model to optimise the cost taking into account a trade-off between good reconstruction and successful achievement of the goal set by the regularising term. The properties imposed by this term include sparsity of the representation, size of the derivative of the representation, and robustness to noise or missing inputs [116].

Generative models such as the variational autoencoder and the generative stochastic networks, on the other hand, can learn useful encodings without resorting to regularisation, although they can naturally learn high-capacity, overcomplete representations of the inputs. This is made possible because these models are trained to approximately maximise the probability of the training data, instead of copying it.

Theoretically, the universal approximator theorem guarantees that the autoencoder can represent the identity function along the domain of the data arbitrarily well, provided that it has enough hidden units. Such shallow network, however, does not allow one to force arbitrary constraints such as sparsity [116]. A deeper autoencoder, on the contrary, can approximate any input to code arbitrarily, provided that it has at least one more hidden layer within the encoder and enough hidden units.

Variational autoencoders

As mentioned before, autoencoders are prone to overfitting if they are not regularised, meaning that they might fail when trying to build a meaningful encoded space, because their main goal is to learn the parameters that best reconstruct the input data, regardless of the encoding structure.

Variational autoencoders (VAEs) are built in a way so that they encode the inputs into latent variables, that exhibit a meaningful structure. With an architecture similar to the one of a standard autoencoder, it is composed by an encoder and a decoder. However, contrarily to a standard autoencoder, a VAE does not encode the data into points, encoding it into distributions instead, with a process that works as a form of regularization. The steps for training a VAE are the following:

- Encode input data as distributions over the latent space.
- Sample a point from the latent space.
- Reconstruct the data from the sampled point.
- Backpropagate the reconstruction error over the network in order to update the network parameters.

The VAE is a deep generative model composed of a stochastic encoder and decoder, that models the relationship between the input random variable x and the low-dimensional latent random variable z [114].

The marginal distribution over the inputs x and the latent variables z can be obtained with the joint distribution $p_\theta(x, z)$:

$$p_\theta(x) = \int p_\theta(x, z) dz \,, \tag{3.10}$$

where $p_\theta(x)$ is an approximation of the true distribution of the data, $p^*(x)$.

When the distributions of a latent variable model $p_\theta(x, z)$ are parameterised by neural networks, the term deep latent variable model (DLVM) is used. One important feature of DLVMs is that the marginal distribution $p_\theta(x)$ can be very complex regardless of the complexity of the conditional distribution in the directed model, allowing good approximations for potentially complicated underlying distributions $p^*(x)$.

One downside of DLVMs, however, is the intractability of the marginal probability of data under the model, which makes maximum likelihood learning very difficult. This is due to the lack of an analytical solution or an efficient estimator for the integral in equation 3.10, which makes it indifferentiable with respect to its parameters. Considering the relation:

$$p_\theta(z|x) = \frac{p_\theta(x,z)}{p_\theta(x)}\,, \tag{3.11}$$

and the fact that $p_\theta(x,z)$ is efficient to compute, $p_\theta(x)$ could be computed through this relation. However, this is not possible since the posterior $p_\theta(z|x)$ is also intractable in DLVMs.

To address the problem of calculating the posterior, the VAE framework provides a way to efficiently optimize DLVMs along with a corresponding inference model, using an optimiser such as stochastic gradient descent.

The encoder is a parametric inference model with parameters ϕ, trained to learn $q_\phi(z|x)$, an approximation of the intractable true posterior distribution $p_\theta(z|x)$, where ϕ are the parameters of the network (weights and biases) and θ are learned parameters. On the other hand, the decoder is trained to learn an approximation of the posterior distribution $p_\theta(x|z)$.

To evaluate the approximation $q_\phi(z|x)$ of the true posterior, the Kullback-Leibler (KL) divergence $D_{KL}\left(q_\phi(z|x)||p_\theta(z|x)\right)$ is used. Although it cannot be computed directly, it can be minimized by maximizing the sum of the Evidence Lower Bound (ELBO) on the marginal likelihood of the data points x_i. The ELBO for each data point is given by

$$ELBO_i = \mathbb{E}_{q_\phi(z|x_i)}\left[log\, p_\theta(x_i|z)\right] - D_{KL}(q_\phi(z|x_i)||p(z))\,, \tag{3.12}$$

where $p(z)$ is a prior distribution of z.

The loss function used to train the VAE is then given by

$$\mathcal{L} = -\sum_i ELBO_i = -\sum_i \left[\mathbb{E}_{q_\phi(z|x_i)}\left[log\, p_\theta(x_i|z)\right] - D_{KL}(q_\phi(z|x_i)||p(z))\right]\,, \tag{3.13}$$

where the summation is calculated over all images in the training set. The first term in (3.13) is seen as a reconstruction error between the inputs and outputs, whereas the second term is a regulariser that prevents the network from assigning to each input a distribution in a different region of the latent space.

In order to use an optimiser to optimise the ELBO with respect to the parameters θ and ϕ, one needs to compute its gradients with respect to these parameters. Because the gradient of the ELBO with respect to the parameters ϕ is difficult to obtain, a reparameterisation trick is used. For the reparameterisation trick, the random variable $z \sim q_\phi(z|x)$ is represented as a differentiable and invertible transformation of an introduced random variable ϵ, given z and ϕ:

$$z = g(\epsilon, \phi, x) \tag{3.14}$$

where ϵ is a random noise sample $\epsilon \sim p(\epsilon)$. As a result, it can be shown that

$$\widetilde{ELBO}_i = log\, p_\theta(x_i, z) - log\, q_\phi(z|x_i) \tag{3.15}$$

is an estimate of the ELBO of the individual data point [127].

The operations of sampling ϵ, transforming z into $g(\epsilon, \phi, x)$ and calculating $\widetilde{ELBO}_i$ are differentiable with respect to the parameters θ and ϕ, and the resulting gradient is used to optimize the ELBO using SGD or other optimisation algorithm, with minibatches of data.

β-VAE

β-VAE [128] is a modification introduced to VAEs with the goal of discovering disentangled latent factors among the data. It does so by introducing an hyperparameter β that regularises the contribution of the KL divergence term in the loss function used to train the VAE. The resulting loss function is the following:

$$\mathcal{L} = -\sum_i ELBO_i = -\sum_i \left[\mathbb{E}_{q_\phi(z|x_i)}\left[log\, p_\theta(x_i|z)\right] - \beta\, D_{KL}(q_\phi(z|x_i)||p(z))\right] \ , \tag{3.16}$$

Evidently, a $\beta - VAE$ with $\beta = 1$ is a regular VAE. When this parameter is increased above 1, it encourages the model to learn a more efficient latent representation of the data, that is disentangled if it contains some underlying factors of variation that are independent. One resulting caveat is that the model is pushed to focus more on the distributions learnt, creating a trade-off between reconstruction ability and disentanglement quality.

3.1.4 Principal component analysis

The principal component analysis is an unsupervised learning algorithm that learns a compressed representation of the data, used in applications such as dimensionality reduction, lossy data compression, feature extraction and data visualisation. It can be defined as the orthogonal projection that maximises the variance of the projected data, onto a lower dimensional linear space called principal subspace [129]. It can also be defined as the linear projection that minimises the mean squared distance between the data points and their projections.

PCA provides a way to examine very high-dimensional data that may not provide very useful information if unprocessed. Each dimension of the data may have only a small fraction of the total information present in the whole dataset, so the PCA algorithm finds a low-dimensional representation of the data that contains as much variation as possible, seeking the observations that vary more along each dimension.

Considering an $n \times p$ data matrix X, with n being the number of examples in the data and p the number of features (dimensionality of the original data), the goal of the PCA algorithm is to project these data into a space with dimensionality l, through a set of l p-dimensional coefficients $w_{(k)} = (w_1, \ldots, w_p)_{(k)}$, called principal components, projecting each row vector $x_{(i)}$ of X, to a new vector of principal components given by

$$t_{k(i)} = x_{(i)} \cdot w_{(k)} \,, \tag{3.17}$$

with $i = 1, \dots, n$ and $k = 1, \dots, l$. Each element of t inherits the maximum variance from X and each coefficient vector w is constrained to be a unit vector.

To maximise the variance, the coefficient vector of the first principal component satisfies the following condition:

$$w_{(1)} = \arg\max \left\{ \frac{w^T X^T X w}{w^T w} \right\} \tag{3.18}$$

To further obtain any k-th component, it is necessary to subtract the $k-1$ previously obtained principal components:

$$\hat{X}_k = X - \sum_{s=1}^{k-1} X w_{(s)} w_{(s)}^{\mathrm{T}} \,, \tag{3.19}$$

and then find the coefficients that extract the maximum variance from $\hat{X}_k$:

$$w_{(k)} = \arg\max \left\{ \tfrac{w^T \hat{X}_k^T \hat{X}_k w}{w^T w} \right\}. \tag{3.20}$$

3.1.5 K-means clustering

K-means clustering [130] is an unsupervised algorithm that partitions the available data into K distinct clusters. To apply K-means clustering, the number of cluster K must first be chosen. Then, K different centroids $\mu^{(1),\dots,\mu^{(K)}}$ are initialised with different values, and each data point in the dataset is assigned to the nearest centroid. After this assignment, each centroid is updated to the mean of all the examples that were assigned to it.

The optimisation objective of K-means clustering is to minimise the within-cluster variation, that is, the amount by which the data points within a cluster differ from each other. In other words, the objective is to assign K clusters in a way that ensures that the within-cluster variation, summed over all the K clusters, is as small as possible. A common choice of a metric for this variation is the squared Euclidean distance, resulting in the following optimisation problem:

$$\underset{C_1,\dots,C_K}{\text{minimise}} \left\{ \sum_{k=1}^{K} \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2 \right\}. \tag{3.21}$$

The problem of minimizing 3.21 precisely is very difficult, so an iterative algorithm that finds a local optimum efficiently is used:

1. Randomly assign a number from 1 to K, to each of the observations, each number i corresponding

to the cluster i. This is the initialisation process.

2. Iterate until cluster assignments stop changing:

   (a) Compute the cluster centroids, given by the feature means for all the observations within the cluster.

   (b) Assign each data point to its nearest centroid (with smaller Euclidean distance).

The local minimum found by the algorithm depends on the initialisation values of the centroids, therefore it is usual to run the algorithm several times in order to look for the one that finds the smaller value for the optimisation objective.

3.1.6 Gaussian Mixture Models clustering

Clustering with Gaussian mixtures aims at describing the available data through a linear superposition of a set of K Gaussian distributions: a Gaussian mixture.

This mixture of Gaussians is given by

$$p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k)\,, \tag{3.22}$$

where $\mathcal{N}(x|\mu_k, \Sigma_k)$ is the density of each Gaussian, called a component of the mixture, each having their own mean μ_k and covariance Σ_k. The parameters π_k are called mixing coefficients, and satisfy the condition

$$\sum_{k=1}^{K} \pi_k = 1. \tag{3.23}$$

Let z be a K-dimensional binary random variable, that has a 1-of-k representation in which a particular element z_k is equal to 1, and all other elements are zero-valued, that verifies the condition

$$p(z_k = 1) = \pi_k. \tag{3.24}$$

It can be shown that the conditional probability of z given x, $\gamma(z_k)$, is given by [129]

$$\gamma(z_k) = p(z_k = 1|x) = \frac{\pi_k \mathcal{N}(x|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x|\mu_j, \Sigma_j)}. \tag{3.25}$$

π_k shall be viewed as the prior probability of $z_k = 1$ and $\gamma(z_k)$ as the corresponding posterior probability once x is observed, or the responsibility that component k takes for explaining x.

Having a dataset with N examples with dimension D each, the log of the likelihood function is given by

$$\ln p(X|\pi,\mu,\Sigma) = \sum_{n=1}^{N} \ln \left\{ \sum_{k=1}^{K} \pi_k \mathcal{N}(x_n|\mu_k,\Sigma_k) \right\}. \tag{3.26}$$

In order to find the optimal parameters for the mixture, one has to differentiate equation 3.26 with respect to the parameters, in order to find the maximum likelihood. There are multiple obstacles faced when trying to maximise 3.26, such as the presence of singularities, so the log likelihood function is not a well posed problem [129]. Another issue arises from the fact that, for any solution, a mixture with K components will have a total of K! equivalent solutions corresponding to the K! ways of assigning K sets of parameters to K components.

A possible approach for finding the solutions for this problem is the Expectation-Maximisation (EM) algorithm [131], that consists in the following steps:

1. Initialise μ_k, Σ_k and π_k and compute the initial value of the log likelihood.

2. Expectation step: Evaluate the responsibilities using the current values of the parameters

$$\gamma(z_{nk}) = \frac{\pi_k \mathcal{N}(x_n|\mu_k,\Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_n|\mu_j,\Sigma_j)} \tag{3.27}$$

3. Maximisation step: Compute new estimates of the parameters using current responsibilities

$$\mu_k^{new} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) x_n \tag{3.28}$$

$$\Sigma_k^{new} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk})(x_n - \mu_k^{new})(x_n - \mu_k^{new})^T \tag{3.29}$$

$$\pi_k^{new} = \frac{N_k}{N} \tag{3.30}$$

where $N_k = \sum_{n=1}^{N} \gamma(z_{nk})$.

4. Evaluate the log likelihood with equation 3.26 and check for convergence.

5. Return to step 2 if convergence criterion not met.

3.2 Approaches

The previous section served as an introduction for the key methods and frameworks that are of major importance to achieve the proposed goal of this thesis. In this section we present our proposed approach for ship detection in SAR images, which is based on two major steps: representation learning and anomaly detection.

3.2.1 Representation learning

The representation learning model is a Convolutional Variational Autoencoder. The encoder is parameterised by a convolutional neural network that has the objective to extract image features. CNNs were chosen for their many advantages and their capabilities when it comes to analyse images, as was detailed in section 3.1.1. The prior distribution over the latent variables, $p(z)$, is defined as an isotropic multivariate Normal distribution, $N(0; I)$. The parameters μ_z and σ_z of the approximate posterior distribution $\sim q(z|x)$ are derived from the final encoder layers, that will output a set of means and log variances. Each mean and variance corresponds to a generated latent distribution. Finally, a sampling layer is added to perform the reparameterisation trick, as follows:

$$z = \mu + \sigma \odot \epsilon, \tag{3.31}$$

where $\epsilon \in N(0; I)$ is an auxiliary noise variable and $\odot$ denotes an element-wise product.

The decoder is a transposed convolutional neural network, symmetric to the encoder CNN, that receives as input a sample z, drawn from the approximate posterior, and outputs an image of the same size as the input images.

The loss function used to train the VAE is composed of two elements: the KL divergence term and the reconstruction error. The reconstruction error can be seen as a term that penalizes the model for not faithfully reconstructing the inputs, which contributes to the maximization of the reconstruction likelihood. The KL divergence term can be seen as a way of penalizing the model for not learning a distribution $q(z|x)$ similar to the true posterior $p(z|x)$. In order to tweak the contribution of each term, the KL divergence term is multiplied by an hyperparameter β. If we increase β we encourage the network to learn a better distribution, whereas if we decrease it, we encourage the network to focus more on the reconstruction task. With the loss function set, the model is then trained using a proper optimisation algorithm, such as Adam or SGD, with minibatches of SAR images without ships. One should notice that this is a totally unsupervised learning method, since no labels are used during training.

A diagram with the overall architecture of the VAE framework is shown in Figure 3.7

3.2.2 Anomaly detection

The anomaly detection strategy is based on the following idea. The representation learning model (the Convolutional VAE) is trained on data without ships, so that it learns the normal patterns of images without ships. At test time, the representation learning model will receive as input images with and without ships and will map those images with different patterns into different regions of the space. Therefore, it is possible to use those representations to distinguish between images with and without ships. For this purpose we used clustering in the approximate posterior mean space μ_z.

The goal is to find the two clusters in this space that best describe the normal data (images without ships) and the anomalous data (images with ships). To achieve this, we propose the usage of the K-means algorithm and Gaussian mixture models to assign the largest cluster to normal images and the smallest to anomalous ones.

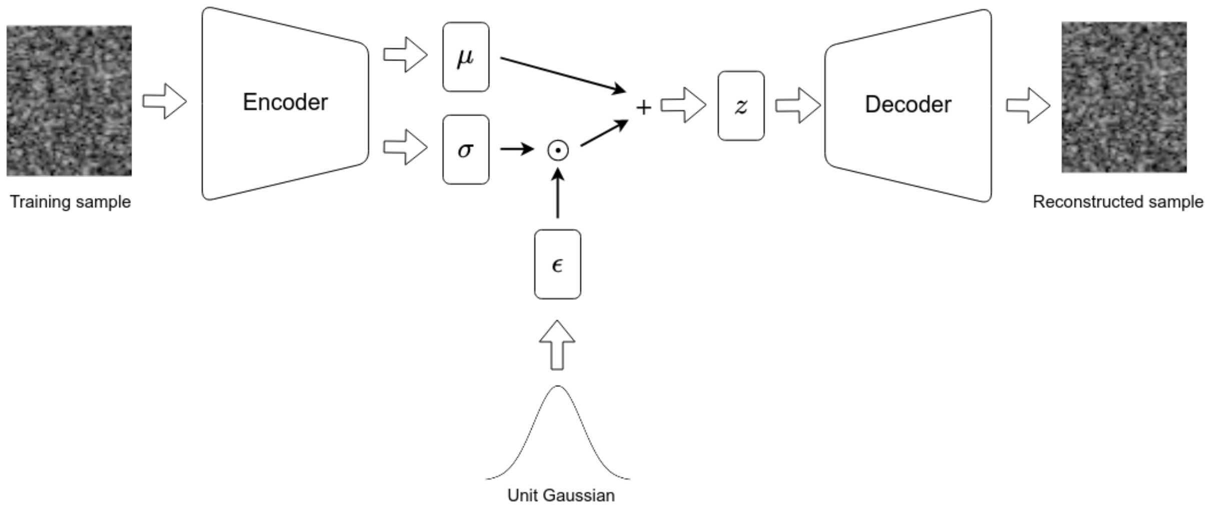


Figure 3.7: General architecture of the VAE.

An overview of the clustering pipeline is shown in Figure 3.8.

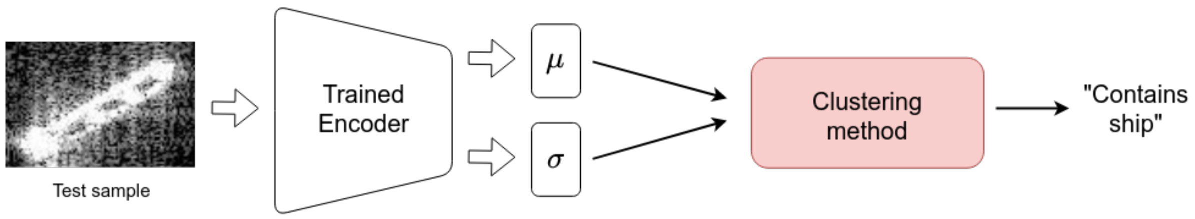


Figure 3.8: Clustering pipeline.

Chapter 4

Implementation

In this chapter, the implementation details of our proposed approach are presented. First, the main characteristics of the dataset are discussed, as well as the processing it had to undergo before being utilized in our experiments. Subsequently, all the details of the whole process required to achieve the final results, from model training to testing are presented.

4.1 Dataset

Deep learning models are highly influenced by the quality and amount of the data used to train it. Usually, the more data available the better the generalisation power of the network, and the lack of annotated data is often a bottleneck in neural networks training.

Although the proposed method does not need labels to be trained, labelled data is still needed to test the final model.

The dataset used throughout this work was made available by Wang et al. [132], and consists of 43,819 256x256 SAR images, all of them containing one or more ships, extracted from 102 Chinese Gaofen-3 images and 108 Sentinel-1 images. An important feature of this dataset is its variety in several settings, such as polarisation, resolution, incidence angle, imaging mode and background complexity, making it a very promising tool to train robust models, with high generalisation power. This is of paramount importance to have a model that performs well on previously unseen data.

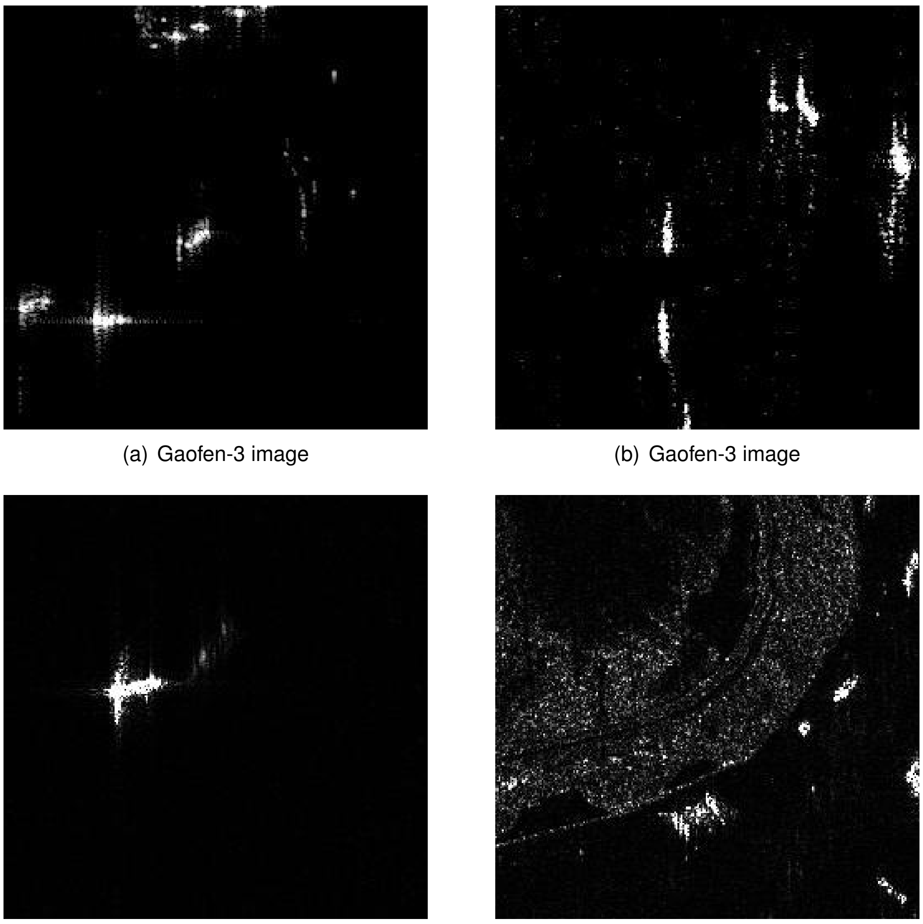
4.1.1 Dataset details

The 102 Gaofen-3 images have resolutions ranging from 3m to 10m, with Ultrafine Strip-Map (UFS), Fine Strip-Map 1 (FSI), Full Polarisation 1 (QPSI), Full Polarisation 2 (QPSII), and Fine Strip-Map 2 (FSII) imaging modes, respectively. For the Sentinel-1 images, imaging modes include S3 Strip-Map (SM), S6 SM, and IW-mode. The properties of the data are briefly described in Table 4.1. Figure 4.1 contains some samples of the aforementioned images, with two examples obtained by the Gaofen-3 satellite and two examples obtained with the Sentinel-1.

The details of the procedure conducted to achieve the final set of images are described in [132].

Sensor	Imaging Mode	Resolution (m)	Swath (km)	Incident angle (°)	Polarisation	#images
GF-3	UFS	3x3	30	20 50	Single	12
GF-3	FS1	5x5	50	19 50	Dual	10
GF-3	QPSI	8x8	30	20 41	Full	5
GF-3	FSII	10x10	100	19 50	Dual	15
GF-3	QPSII	25x25	40	20 38	Full	5
Sentinel-1	SM	1.7x4.3 to 3.6x4.9	80	20 45	Dual	49
Sentinel-1	IW	20x22	250	29 46	Dual	10

Table 4.1: Details of the dataset. From [132].


(a) Gaofen-3 image (b) Gaofen-3 image

(c) Sentinel-1 image (d) Sentinel-1 image

Figure 4.1: Images from the original dataset [132].

Each image file in the original data is accompanied with an Extensible Markup Language (XML) file, that contains the pixel coordinates of all the bounding boxes that enclose each ship for the respective image.

4.1.2 Pre-processing

To build a no-ship image dataset, we randomly cropped portions of these images, all of which contained no ships, which was possible since the ships location and size were labelled. This process resulted in a total of 43,789 images without ships - 42,789 images for the training set and 1,000 images for the test set - all with varying width, height, and aspect ratio. To complete the test set, 500 images with ships were cropped using the ship labels, resulting in a test set composed of 1,500 images: 500 with ships and 1,000 without ships. For this purpose, a python script was written to perform these steps automatically. Figure 4.2 contains some examples of the resulting dataset.

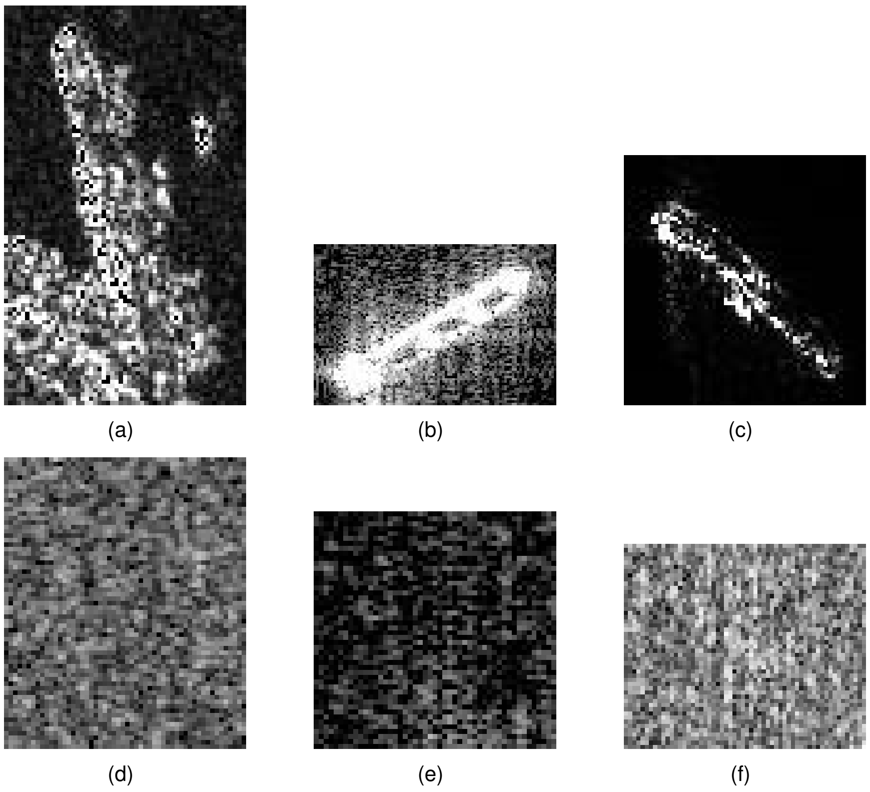


Figure 4.2: Examples of images with ships (a, b, c), and without ships (d, e, f), from the resulting dataset.

As will become clear later, the model used requires a constant image input size, so each image is appropriately resized to the desired dimensions before entering the network. Moreover, since the image pixel values are, by default, in a range from 0 to 255, each image pixel is rescaled to a 0-1 range, dividing each by 255. This step is common practice in neural networks training.

4.2 Representation learning

The first part of the experimental setup consisted in building the VAE network. As stated before, the job of the VAE is to extract features relevant enough so that the resulting low-dimensional latent space for each sample provides sufficient information for the clustering algorithms to assign each image to its respective cluster. Hence, assuring that a good model is built is of paramount importance for the success of this work.

Since literature on the usage of VAEs to extract features from SAR images was lacking, the process of choosing the right architecture and parameters was very empirical, in the sense that many tests with different configurations had to be conducted in order to arrive at a desirable one, chosen according to some metric. The choices that had to be done included the following:

- Number of layers and their types (pooling layer, convolutional layer, FC layer, etc.)
- Layer specifications
  - Number of filters, kernel size, strides and padding (in the case of convolutional layers)
  - Pooling size (for pooling layers)
  - Number of hidden units (in the case of FC layers)
- Optimiser
- Learning rate
- Latent space dimensions
- The value of β
- Reconstruction error function
- Batch size

Naturally, the number of configurations that we are able to try is limited by the available resources, and there is no guarantee that the ideal configuration is among the set of configurations chosen for training. Nevertheless, it is expected that among the possible configurations in this set is a configuration that performs well and achieves satisfying results.

4.2.1 VAE architecture

The chosen architecture for the encoder part of the network is shown in table 4.2, with each layer's specifications. Note that this table does not include the reshaping/flattening steps.

Index	Type	Kernel size	Filters	Stride	Padding	Activation	Output shape
1	Conv2D	(3,3)	16	1	Same	ReLU	(56,56,16)
2	MaxPool2D	(2,2)	-	2	Valid	Linear	(28,28,16)
3	Conv2D	(3,3)	32	1	Same	ReLU	(28,28,32)
4	MaxPool2D	(2,2)	-	2	Valid	Linear	(14,14,32)
5	Conv2D	(3,3)	64	1	Same	ReLU	(14,14,64)
6	MaxPool2D	(2,2)	-	2	Valid	Linear	(7,7,64)
7	Dense	-	d_z	-	-	Linear	(d_z)
8	Dense	-	d_z	-	-	Linear	(d_z)
9	Custom	-	-	-	-	-	(d_z)

Table 4.2: Encoder architecture.

Conv2D corresponds to the convolutional layer, whereas MaxPool2D corresponds to the max pooling operation, and d_z is the desired size of the latent space. The outputs of layer 6 are wired to both layers 7 and 8, which output the sets of means and log variances, respectively. Layer 9 is a custom layer (or lambda layer) that performs the reparameterization trick to sample a point z from the latent space, that is assumed to generate the input image, and will feed it to the decoder part of the network. This sampling layer should be regarded as an intermediate layer between the encoder and the decoder, cleverly placed to allow the usage of gradient descent over the network. Figure 4.3 depicts an overview of the data pipeline of the encoder, for a latent space with dimensions $d_z = 256$.

In order to avoid misinterpretations and for clarification, z is referred to as a point in the sense that it is a data point, provided with d_z dimensions.

After sampling from the latent space, the point z is fed to the decoder, that attempts to reconstruct the original input image based on the sample. The architecture details of the decoder part of the network is shown in table 4.3, excluding the reshaping/flattening steps. Conv2DTranspose layers are convolutional layers that also learn the best way to upsample their inputs, although here the dimensions are kept constant. The upsampling job is conducted by the UpSampling2D layers, using nearest-neighbor upsampling to scale the image up.

Index	Type	Kernel size	Filters	Stride	Padding	Activation	Output shape
1	Dense	-	3136	-	-	ReLU	(3136)
2	Conv2DTranspose	(3,3)	64	1	Same	ReLU	(7,7,64)
3	UpSampling2D	(2,2)	-	2	-	Linear	(14,14,64)
4	Conv2DTranspose	(3,3)	32	1	Same	ReLU	(14,14,32)
5	UpSampling2D	(2,2)	-	2	-	Linear	(28,28,32)
6	Conv2DTranspose	(3,3)	16	1	Same	ReLU	(28,28,16)
7	UpSampling2D	(2,2)	-	2	-	Linear	(56,56,16)
8	Conv2DTranspose	(3,3)	1	1	Same	Sigmoid	(56,56,1)

Table 4.3: Decoder architecture.

Figure 4.4 depicts an overview of the data pipeline of the decoder, for a latent space with dimensions $d_z = 256$. The InputLayer simply corresponds to the sampled point z, with dimensions d_z. One should note that for the last layer, the activation function implemented is the sigmoid, since we wish to output an image with pixel values between 0 and 1.

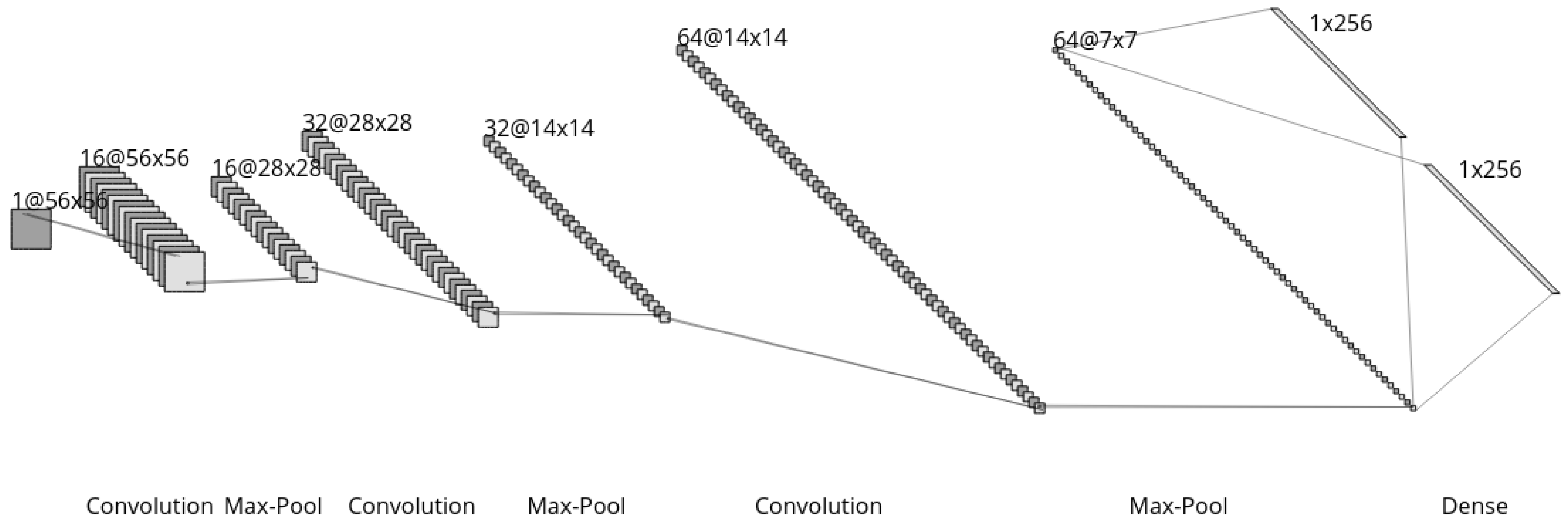


Figure 4.3: Encoder architecture overview.

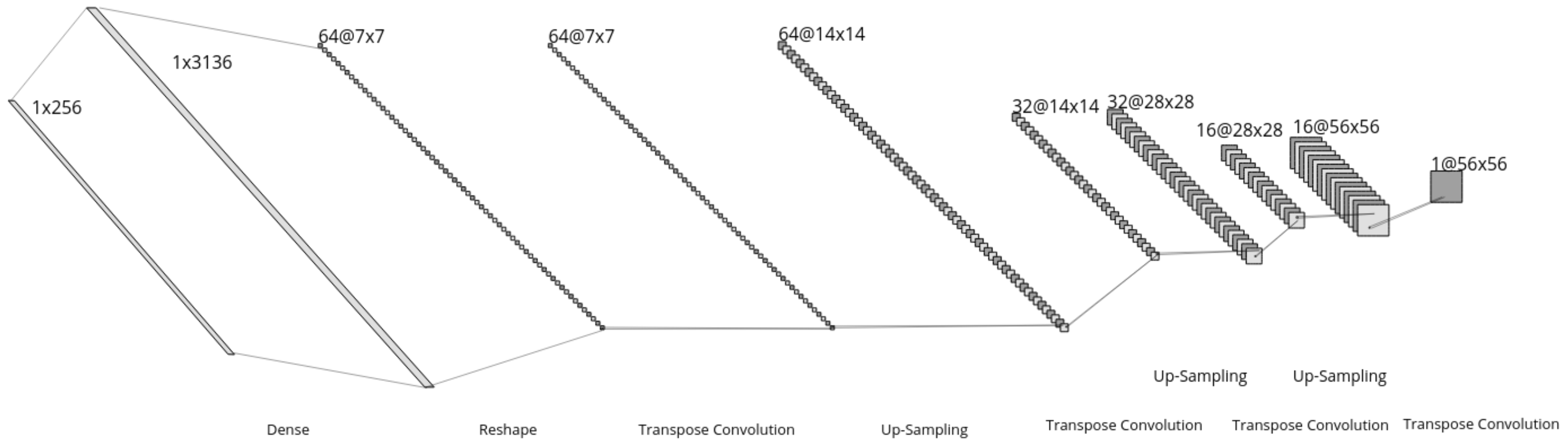


Figure 4.4: Decoder architecture overview.

4.2.2 VAE training

With the overall architecture of the model established, there are still some hyperparameters to define in order to begin training and testing the network.

Although batch normalization is a widely used technique to train deep learning models, we opted not to implement it since the added stochasticity along with the stochasticity from sampling may cause some instability, so it is common practice to avoid it when training VAEs [133].

For the optimiser, Adam was chosen with a learning rate of $lr = 3 \times 10^{-4}$ and a mini-batch size of 128. The reconstruction error function chosen was the mean squared error (MSE).

In order to study the influence of varying the penalty associated with the KL divergence term, the network is trained several times with different values of β. In the end, it is expected that the set of different results for each value of β, evaluated on the test set, will allow us to draw some conclusions about the behaviour of the representation learning task in situations in which optimisation focuses more on the reconstruction versus situations in which the optimisation objective is more oriented towards learning a good distribution for the data. The goal here is to train enough different models to allow us to determine what is the best balance between these two terms.

Additionally, for each different value of β, the network is trained with multiple values for the latent space dimensions, in order to analyse how the dimensionality of the generated space influences the results of the model. Each dimension of this compressed representation is expected to reflect, to some extent, some feature that characterizes the training data. In addition, it is also expected that the bigger these dimensions are, the better the representation learned by the network, although a too big latent space may lead the model to overfit the training set.

Table 4.4 summarises the set of hyperparameters that were used in each training routine.

Optimiser	lr	β	d_z	Loss	Batch
Adam	3×10^{-4}	$\{0, 0.3, 0.7, 1, 2, 4, 10\}$	$\{4, 16, 64, 128, 256\}$	MSE+KL Divergence	128

Table 4.4: Training hyperparameters.

Accurately predicting the number epochs that the network would need to find the best parameters is not possible, so early stopping was used to find them, and avoid overfitting and underfitting of the network to the training set. Therefore, model performance on the validation set was monitored during training, namely the loss function value over time, and the weights that gave the best results stored.

An epoch number limit of 300 was imposed, and stopping triggered if there had not been any improvement in the loss value in the past 50 epochs. 25% of the training data was separated to be used for validation.

4.3 Anomaly detection

After the aforementioned process, we had our model trained with 35 different configurations of hyperparameters, ready to be tested with the test set of 1500 images with and without ships, and try to cluster them based on the features extracted by the resulting trained encoder. For this task, two clustering methods were used to automatically separate our data into two groups. All the implementation steps performed to achieve this goal are explained in this section.

4.3.1 Data visualisation

As was shown in previous sections, each input image will generate a set of d_z means and a set of d_z variances, with d_z ranging from 4 to 256. Since visualising each dimension of such high-dimensional data would be infeasible, we opted to employ PCA with two components to the generated space, in order to visualise the resulting data in two dimensions.

4.3.2 Clustering in the generated space

After training, the 1500 images put aside for testing are passed through each trained encoder to generate their corresponding sets of means and variances. Then, GMMs clustering with $n_{components} = 2$ and K-means clustering with $n_{clusters} = 2$ are used to split the data into two clusters. The smallest cluster assigned by these algorithms is categorized as a cluster of images with ships (anomalies), whereas the biggest cluster is labelled as a cluster of images without ships, or normal images.

4.4 Results evaluation

To evaluate the performance of the framework, a set of evaluation metrics are used to compare the labels assigned by the clustering algorithms with the original labels of the test set. Moreover, a CNN with an architecture similar to the encoder is built and trained with the 42789 images used for training the VAE, plus 58536 images with ships cropped from the original dataset, in order to compare the performance of a supervised model with the proposed unsupervised framework.

Evaluation metrics

Before explaining the evaluation metrics used to evaluate the performance of the model, the following terms regarding the labels assigned by the algorithm must be introduced:

- True negatives: data examples with negative labels (ground truth) that are predicted as negative by the algorithm that is being tested
- True positives: data examples with positive labels that are predicted as positive by the algorithm that is being tested
- False negatives: data examples with positive labels that are predicted as negative by the algorithm that is being tested
- False positives: data examples with negative labels that are predicted as positive by the algorithm that is being tested

These are the values that compose the confusion matrix, and are needed to compute the metrics used to evaluate the clustering results: precision, recall, accuracy and F1-score. The precision is given by

$$Precision = \frac{True\ positives}{True\ positives + False\ positives}, \tag{4.1}$$

and should be used as a metric when the objective is to limit the number of false positives.

Recall is used when it is important to avoid false negatives, and is given by

$$Recall = \frac{True\ positives}{True\ positives + False\ negatives}. \tag{4.2}$$

Accuracy measures the rate of correct predictions:

$$Accuracy = \frac{True\ positives + True\ negatives}{Number\ of\ samples}. \tag{4.3}$$

The F1-score offers a way to summarise both precision and recall, penalising the model for either having bad precision or bad recall:

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}. \tag{4.4}$$

These four metrics are used to evaluate the performance of the model on the test set, for each set of parameters.

Comparing the results with a CNN

As mentioned before, a CNN is trained with a dataset of 42789 images without ships (also used to train the VAE), plus 58536 images with ships, in order to compare its results with the ones obtained with the proposed framework. The layers used in the CNN are detailed in table 4.5.

Index	Type	Kernel size	Filters	Stride	Padding	Activation	Output shape
1	Conv2D	(3,3)	16	1	Same	ReLU	(56,56,16)
2	MaxPool2D	(2,2)	-	2	Valid	Linear	(28,28,16)
3	Conv2D	(3,3)	32	1	Same	ReLU	(28,28,32)
4	MaxPool2D	(2,2)	-	2	Valid	Linear	(14,14,32)
5	Conv2D	(3,3)	64	1	Same	ReLU	(14,14,64)
6	MaxPool2D	(2,2)	-	2	Valid	Linear	(7,7,64)
7	Dense	-	16	-	-	ReLU	(16)
8	Dense	-	1	-	-	Sigmoid	(1)

Table 4.5: CNN architecture.

This network was trained with an Adam optimiser, with a learning rate of $lr = 1 \times 10^{-4}$, using binary crossentropy as the loss function, and a batch size of 128. 25% of the training set was put aside for validation, used to monitor the performance of the network and to trigger early stopping when there were no improvements on the validation accuracy. An overview of the architecture of the implemented CNN is depicted in figure 4.5.

Afterwards, the evaluation metrics mentioned above were calculated for both the CNN and the proposed model, and compared with each other. The objective of this comparison is to see how close the performance of the unsupervised method can get to the performance of a supervised method with similar characteristics and trained under similar circumstances.

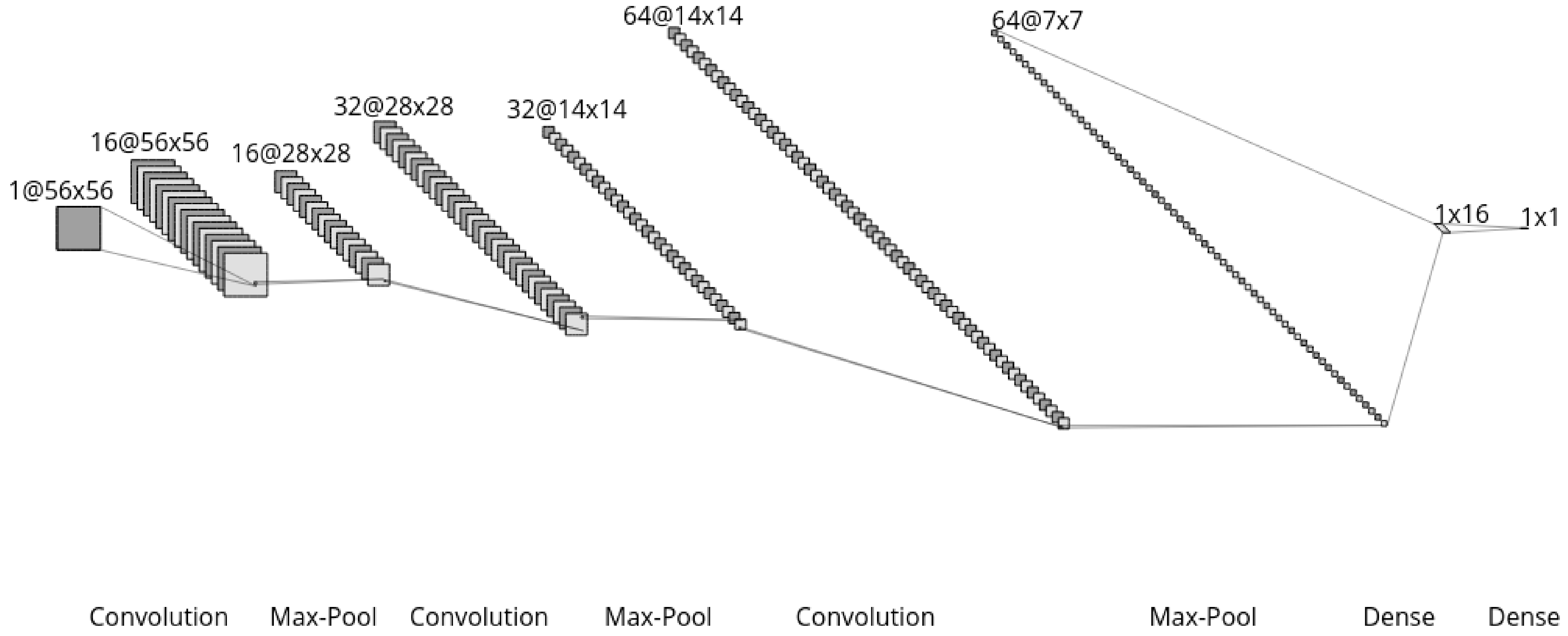


Figure 4.5: CNN architecture overview.

Chapter 5

Results

In this chapter, the results of the proposed method are presented. We start by presenting the training results for the different sets of parameters, followed by a discussion on the clustering of the test set, using two different clustering methods. This discussion will primarily focus on the influence of the variation of the parameters in the final outcome of the model.

The algorithms were implemented using python and the Keras [134] library, with TensorFlow [135] backend. The specifications of the laptop used to train the model were the following:

- CPU: Intel Core i5-7300HQ @ 2.50GHz
- GPU: NVIDIA GeForce GTX 1050 Ti 4GB
- RAM: 8GB
- OS: Ubuntu 18.04.3 LTS

5.1 VAE training

As previously explained in Chapter 4, the VAE was trained with early stopping, with the patience parameter (maximum number of epochs elapsed without improvement of the validation error) set to 50, and maximum number of epochs set to 300. All the training cycles (for each set of parameters) combined took about 62552 seconds to finish, resulting in an approximate training time of 17h 30m.

Figures 5.1 to 5.5 depict the evolution of the validation loss over number of epochs during training for each value of d_z. Each of these figures corresponds to the evolution of the validation loss for each value of β in the set $\{0,\, 0.3,\, 0.7,\, 1,\, 2,\, 4,\, 10\}$, respectively.

Looking at the results shown in these figures, it is possible to observe that generally, regardless of the size of the latent space, the model converges to lower loss values for lower values of beta. This is expected since increasing β leads to increasing the KL divergence term in the loss function. Moreover, bigger β values reveal a tendency to delay the time of convergence of the model. Both of these two observations can possibly be justified by the constraints imposed by $\beta > 1$. As explained in Section 3.1.3, increasing β above 1 limits the capacity in the latent information channel, pushing the model to

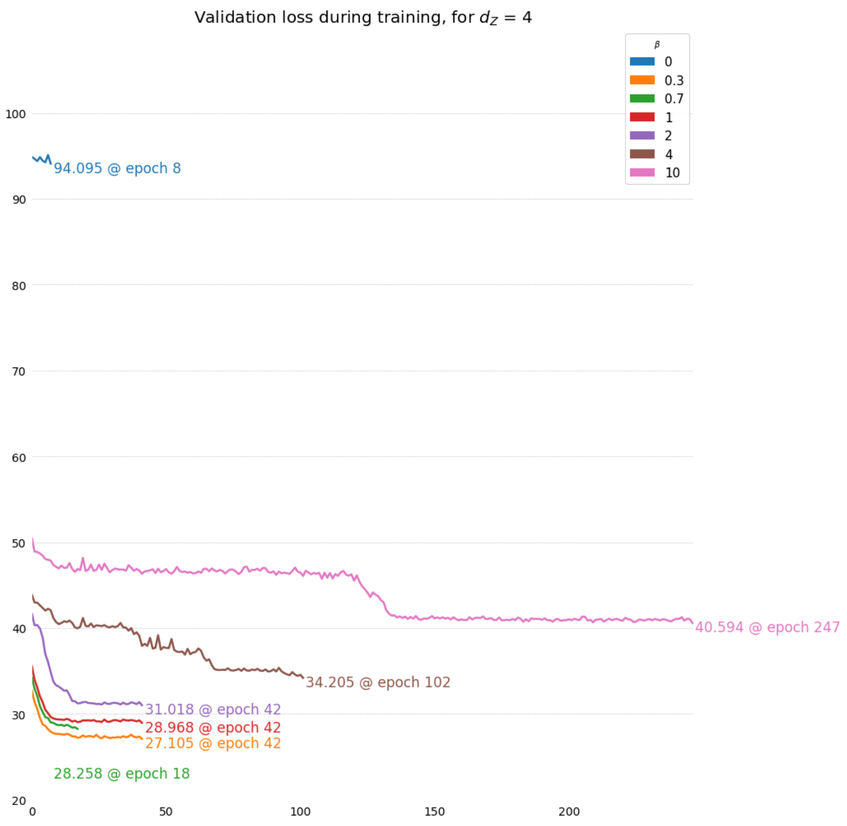


Figure 5.1: Total validation loss over number of epochs for $d_z = 4$.

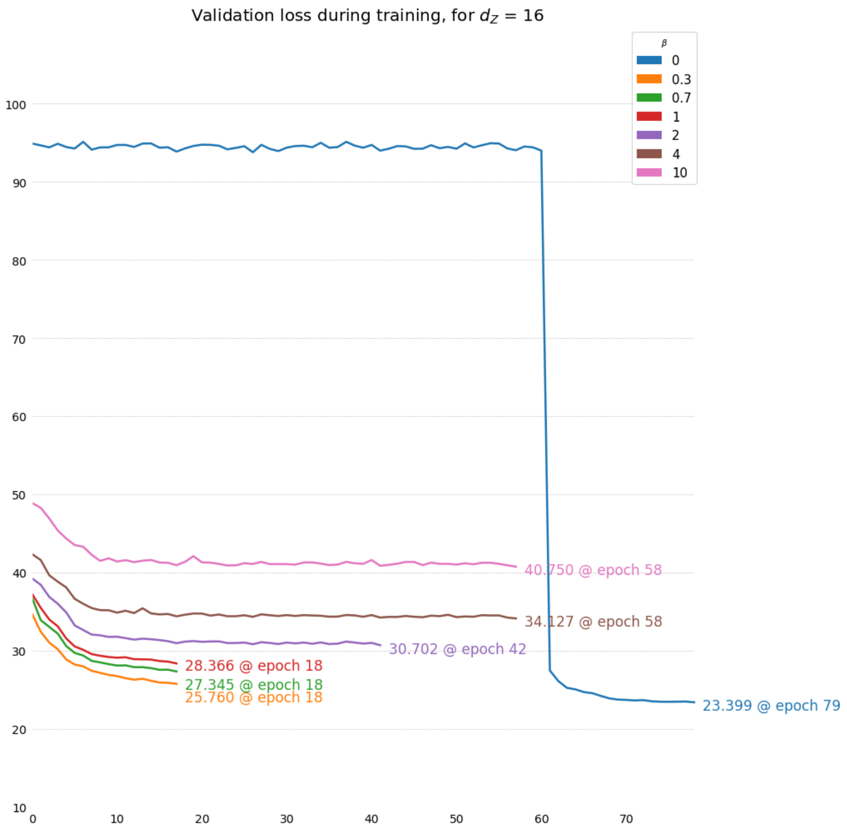


Figure 5.2: Total validation loss over number of epochs for $d_z = 16$.

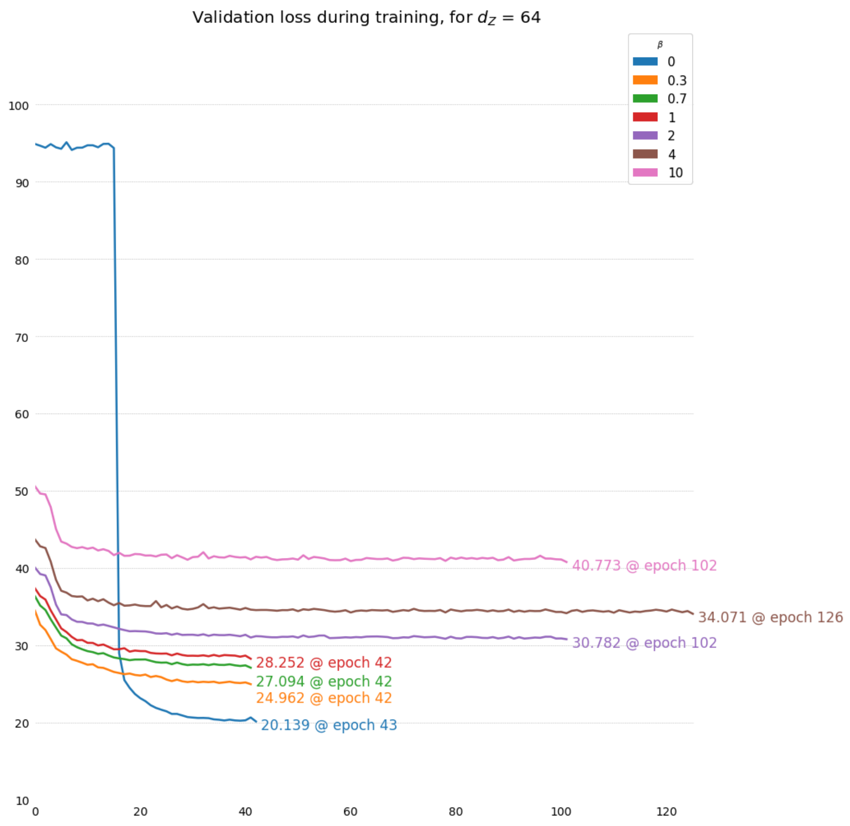


Figure 5.3: Total validation loss over number of epochs for $d_z = 64$.

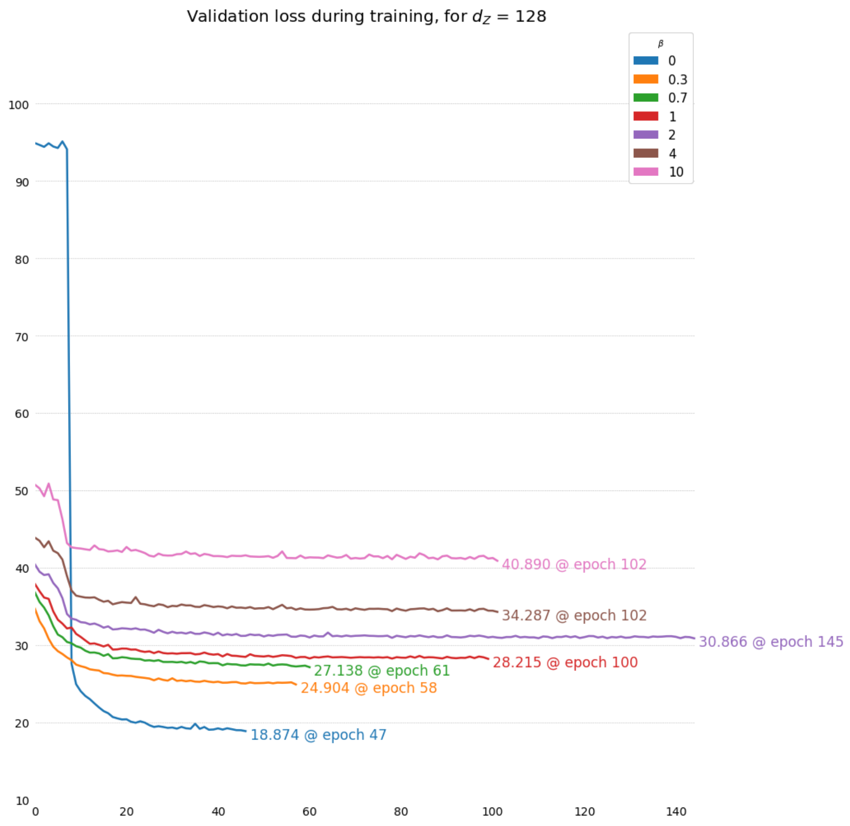


Figure 5.4: Total validation loss over number of epochs for $d_z = 128$.

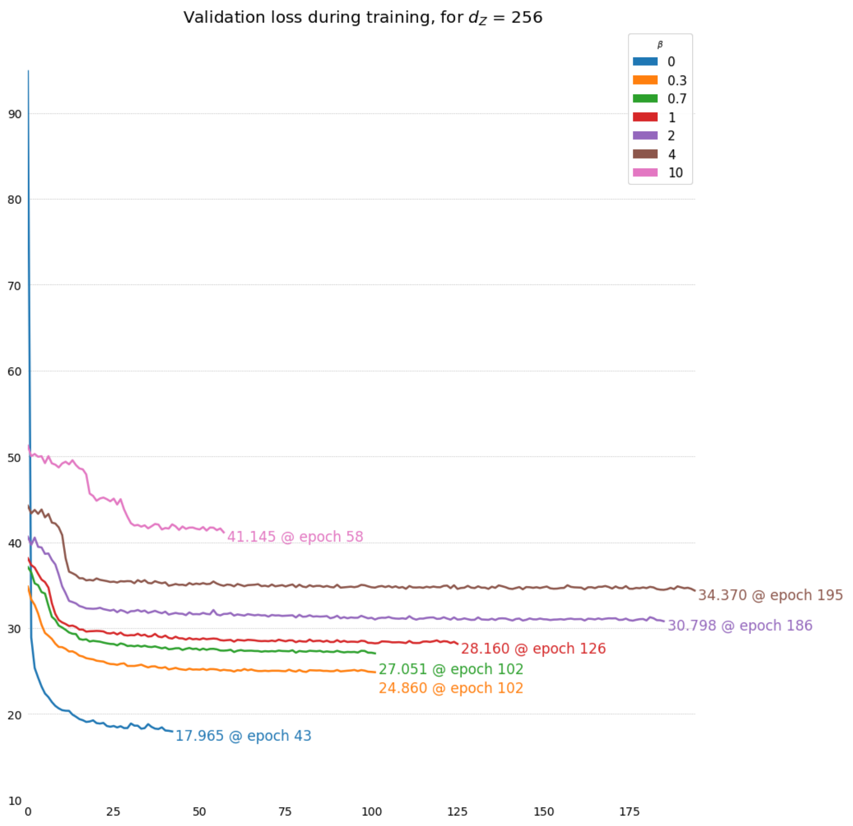


Figure 5.5: Total validation loss over number of epochs for $d_z = 256$.

learn a more efficient representation of the data, while looking for independent factors of variation, and trading off some information preservation (reconstruction quality). Naturally, this leads to higher values in the reconstruction term of the loss function.

Inspecting the final loss values for each combination of parameters, it is also possible to observe that for values of β smaller than 1, the convergence loss value tends to decrease with the increase of d_z, possibly due to the fact that a bigger latent space allows better representations of the inputs, resulting in a smaller reconstruction error. As β increases, this effect becomes less noticeable, possibly due to the limitations imposed by the effect mentioned above.

5.2 Results of anomaly detection on the test set

In this section, we perform an evaluation of the framework in terms of its ability to detect anomalies in the test set.

We start by presenting the results of applying the chosen clustering methods to the generated mean space, which reflect the quality of the VAE's encoding capabilities. Then, we analyse the generated log-variance space, as well as the reconstruction errors associated with the test images.

5.2.1 Clustering in the approximate posterior mean space

This section presents the results obtained by applying the two chosen clustering algorithms to the 35 mean spaces generated by the trained models. One should notice that, throughout this section, we often refer to the results of applying the algorithms to the mean space of a given model simply as the results of the model, in order to avoid unnecessary repetitions.

K-means

Table 5.1 presents the accuracy of the models for every pair of β and d_z, using K-means clustering in the approximate posterior mean space. The highest accuracy (lowest error rate) for each value of β is highlighted in bold, whereas the best accuracy values for each value of latent dimensions d_z are underlined.

		Accuracy (%)				
		$d_z = 4$	$d_z = 16$	$d_z = 64$	$d_z = 128$	$d_z = 256$
β	0	**82.80**	64.67	66.20	63.73	63.40
	0.3	<u>93.80</u>	<u>95.67</u>	<u>95.60</u>	<u>96.20</u>	<u>**96.53**</u>
	0.7	92.67	94.87	95.33	95.60	**95.87**
	1	93.73	93.87	94.60	94.53	**95.07**
	2	92.47	**94.07**	93.13	92.13	92.80
	4	73.20	**92.93**	91.60	90.40	89.60
	10	76.33	76.87	**77.53**	77.33	75.60

Table 5.1: Accuracy of anomaly detection obtained with K-means using generated mean space.

Inspecting table 5.1, it is evident that clustering had the most success using the space generated by the model trained with $\beta = 0.3$, outperforming all the other models trained with different values of β in terms of accuracy. In order to make it easier to interpret the overall impact of the variation of this parameter in the error rate of the model, a plot of accuracy over β is shown in figure 5.6.

Some conclusions can be drawn from this figure. As stated above, accuracy peaks at $\beta = 0.3$ for all values of d_z, with the highest value of all being the one obtained with $d_z = 256$. Increasing β above 0.3 makes the accuracy drop consistently for almost all values of d_z. The effect of decreasing β bellow 0.3 cannot be determined properly, since no intermediate values between this number and 0 were tested. We can, however, conclude that setting it to 0 has a very negative impact on the classification accuracy, since all models trained with this value of β report the highest error rate, except for the one trained with $d_z = 4$.

Further testing could be conducted with values of β within the interval $[0,\ 0.7]$, since there might be a value in this range that leads to better accuracy results.

Going back to table 5.1, it is possible to observe that for values of $\beta = \{0.3,\ 0.7,\ 1\}$, the increase in latent space dimensions promotes an increase in the accuracy, having its highest value for $d_z = 256$. Additional studies could be done with $d_z > 256$, in order to check if there are latent space dimensions

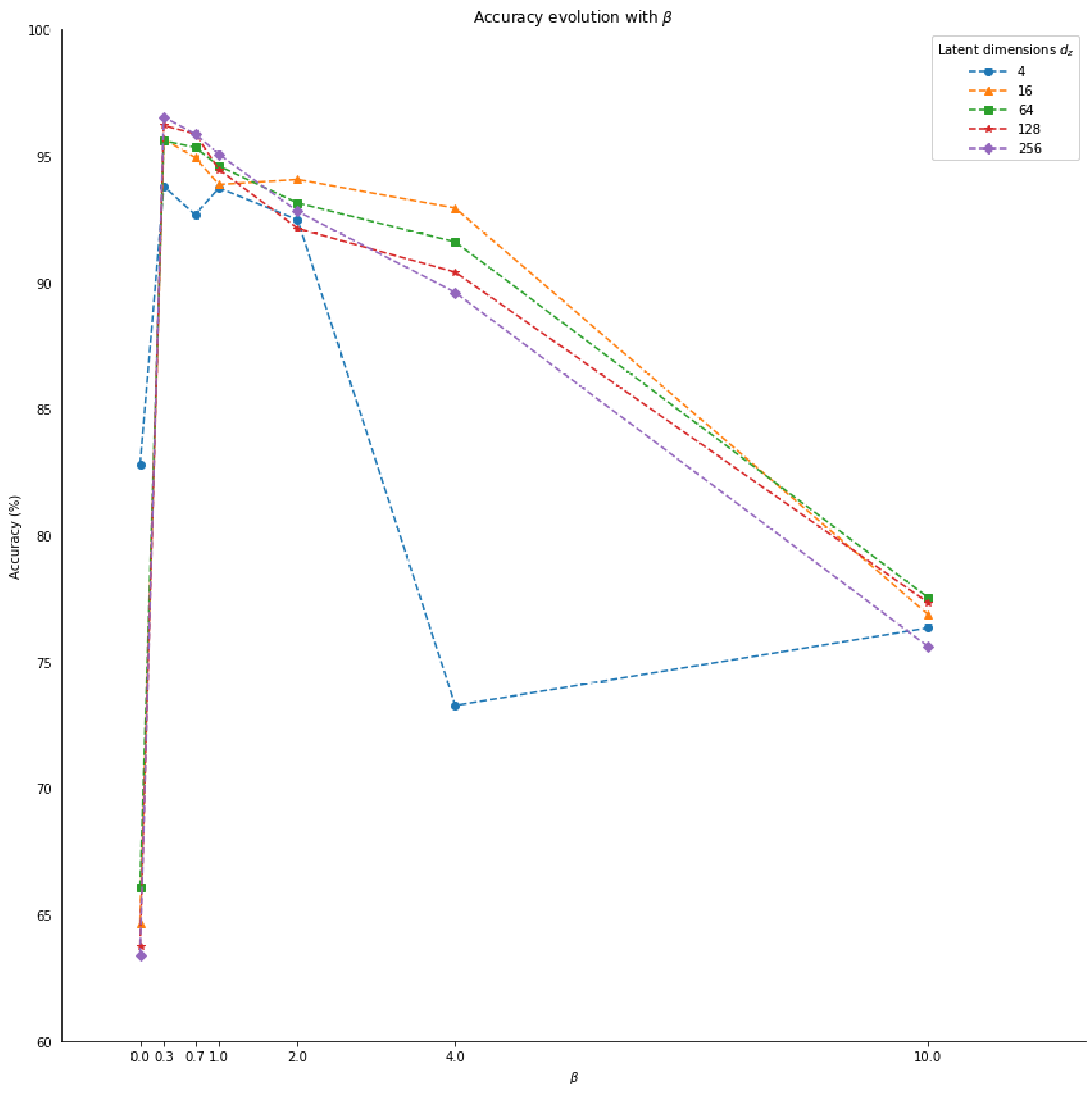


Figure 5.6: Evolution of accuracy over β.

that produce better results, since the obtained values suggest that training with higher dimensions would possibly improve accuracy.

To decide which model works better, other evaluation metrics must be analysed. Table 5.2 contains the F1-scores for the classification using K-means clustering, with the best score highlighted in bold. This is usually a better metric than accuracy if we want to have a good balance between false positives and false negatives, and avoid models that are biased towards one specific class of the data. In addition, precision and recall for each pair of parameters are shown in Tables 5.3 and 5.4, respectively. Evaluating precision will allow us to draw conclusions regarding false positives (false alarms), while recall will allow us to infer about false negatives (missed detections).

Inspecting Tables 5.3 and 5.4, we observe that, generally, precision values are considerably higher than recall values (with the results for $\beta = 0$ and $\beta = 10$ being exceptions), meaning that missed

		F1-score				
		$d_z = 4$	$d_z = 16$	$d_z = 64$	$d_z = 128$	$d_z = 256$
	0	0.7490	0.6187	0.6461	0.6047	0.6070
	0.3	0.8992	0.9321	0.9317	0.9412	**0.9462**
	0.7	0.8791	0.9218	0.9271	0.9317	0.9356
β	1	0.8988	0.9013	0.9143	0.9120	0.9219
	2	0.8762	0.9046	0.8884	0.8700	0.8821
	4	0.6850	0.8853	0.8612	0.8378	0.8207
	10	0.7033	0.7037	0.7087	0.7089	0.6691

Table 5.2: F1-score of anomaly detection obtained with K-means using generated mean space, with the highest score highlighted in bold.

		Precision				
		$d_z = 4$	$d_z = 16$	$d_z = 64$	$d_z = 128$	$d_z = 256$
	0	0.7292	0.4831	0.4978	0.4749	0.4727
	0.3	**0.9811**	0.9759	0.9657	0.9723	0.9808
	0.7	0.9710	0.9776	0.9674	0.9657	0.9740
β	1	0.9743	0.9722	0.9708	0.9707	0.9754
	2	0.9685	0.9746	0.9693	0.9681	0.9712
	4	0.5631	0.9646	0.9583	0.9588	0.9649
	10	0.6029	0.6140	0.6240	0.6198	0.6106

Table 5.3: Precision of anomaly detection obtained with K-means using generated mean space, with the highest value highlighted in bold.

		Recall				
		$d_z = 4$	$d_z = 16$	$d_z = 64$	$d_z = 128$	$d_z = 256$
	0	0.770	0.860	**0.920**	0.832	0.848
	0.3	0.830	0.892	0.900	0.912	0.916
	0.7	0.804	0.872	0.890	0.900	0.900
β	1	0.834	0.840	0.864	0.860	0.874
	2	0.800	0.844	0.820	0.790	0.808
	4	0.874	0.818	0.782	0.744	0.714
	10	0.844	0.824	0.820	0.828	0.740

Table 5.4: Recall of anomaly detection obtained with K-means using generated mean space, with the highest value highlighted in bold.

detections are more prevalent than false alarms among these models. It is important to realise that there are some models that present good recall values when compared to the rest of the group, but have very poor precision values. One example of this is the model trained with $\beta = 0$ and $d_z = 64$, that has a recall value of 0.920 (the highest among all models), but shows one of the lowest precision values (0.4978). This is an indicator that this model is probably labelling the majority of the images as positive. This is confirmed by the normalised confusion matrix of the model in these conditions, depicted in Figure 5.7, showing that 46.4% of the negative images were predicted as positive.

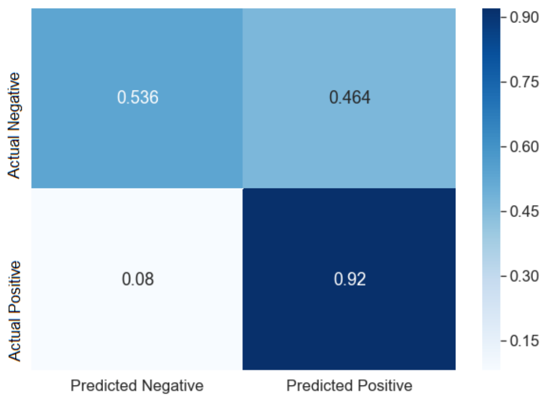


Figure 5.7: Normalised confusion matrix for $\beta = 0$ and $d_z = 64$.

Similarly to what happened with the accuracy values, the models trained with $\beta = 0.3$ outperformed all the others in terms of F1-score, and the one that showed to perform better was the model trained with $d_z = 256$ and $\beta = 0.3$. Therefore, it is fair to conclude that this was the best model, and will be chosen for a deeper analysis later in this section.

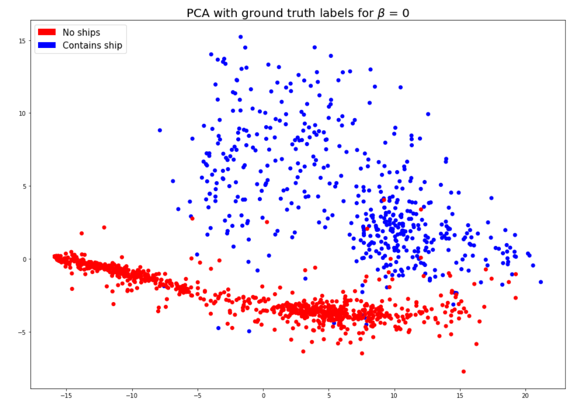
Before proceeding to this analysis, it is also important to visualise the results of applying PCA with 2 components to the generated mean spaces, in order to understand how the encoder distributed each image in the latent space. Figure 5.8 contains seven images corresponding to the 2-component PCA representation of the generated mean space (z_{mean}), with fixed $d_z = 256$ and for $\beta = \{0, 0.3, 0.7, 1, 2, 4, 10\}$.

Each axis represents one dimension of the resulting PCA representation, with the horizontal axis corresponding to the first dimension and the vertical axis to the second dimension.

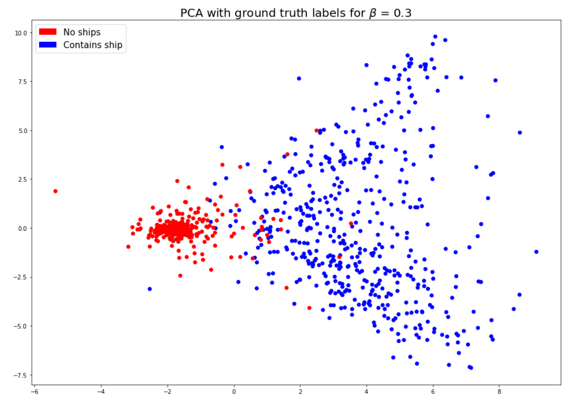
The main idea behind using VAEs to distinguish images with ships from images without ships, was that it was expected that the encoder trained only with normal images (no ships) would learn how to represent them and assign those images a meaningful location in the latent space. In this figure it is possible to observe that this expectation was successfully met for some of the trained models. Namely, in Figures 5.8 (b), (c), (d), and (e), normal images are aggregated together in well defined clusters (red dots in the figures), whereas anomalous images (with ships) are spread across both dimensions in this compressed two-dimensional space (blue dots in the figures). One should notice, however, that in Figure 5.8 (e) there is a clear increase in superposition between data points from both classes when compared to Figures 5.8 (b), (c), and (d), also reflected in the accuracy results obtained.

Interestingly, although we can visually notice an apparent well-defined separation between both classes for $\beta = 0$ (Figure 5.8 (a)), the K-means algorithm was not able to do this separation successfully, showing poor F1-score and accuracy.

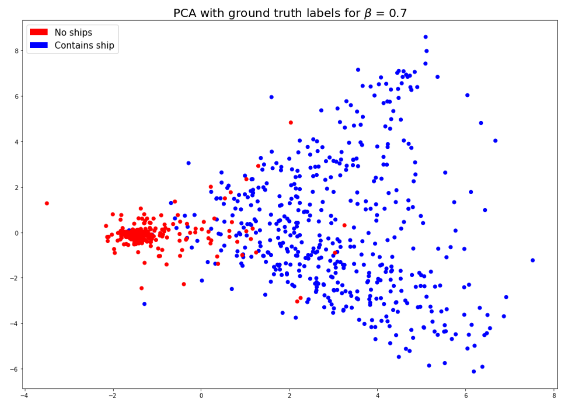
Figure 5.8 (g) confirms that for $\beta = 10$ the model was not able to separate the two classes of data in the latent space, justifying its unsatisfying results in terms of accuracy and F1-score.



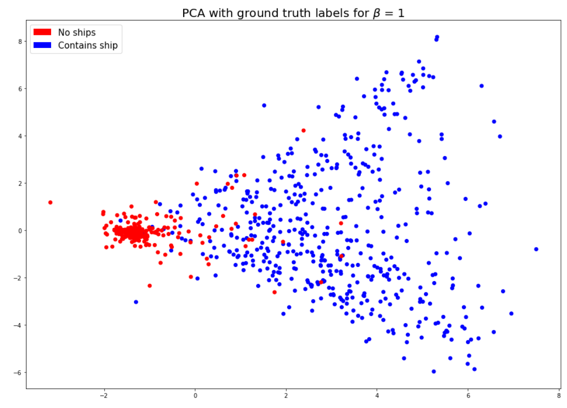
(a)



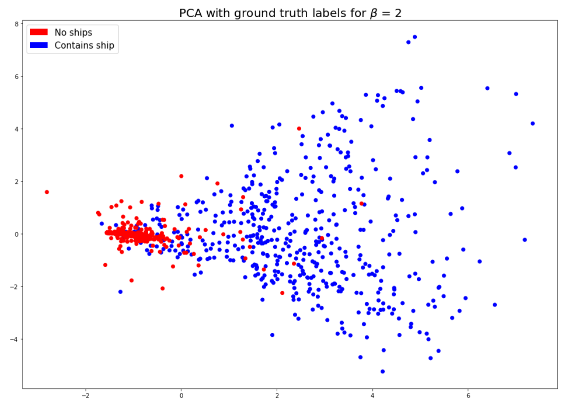
(b)



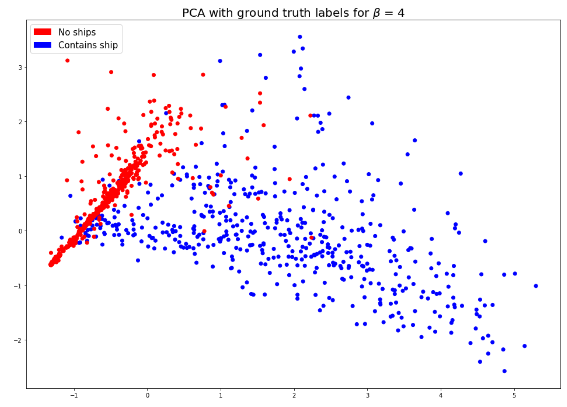
(c)



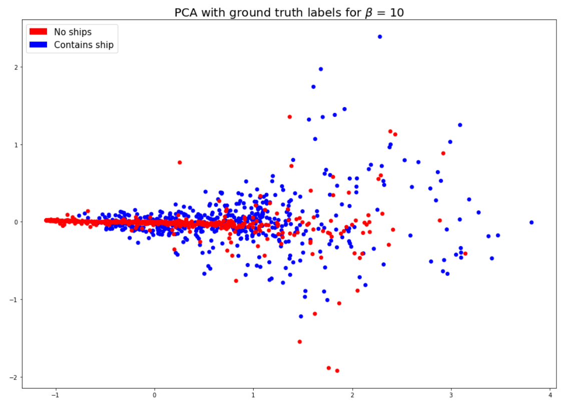
(d)



(e)



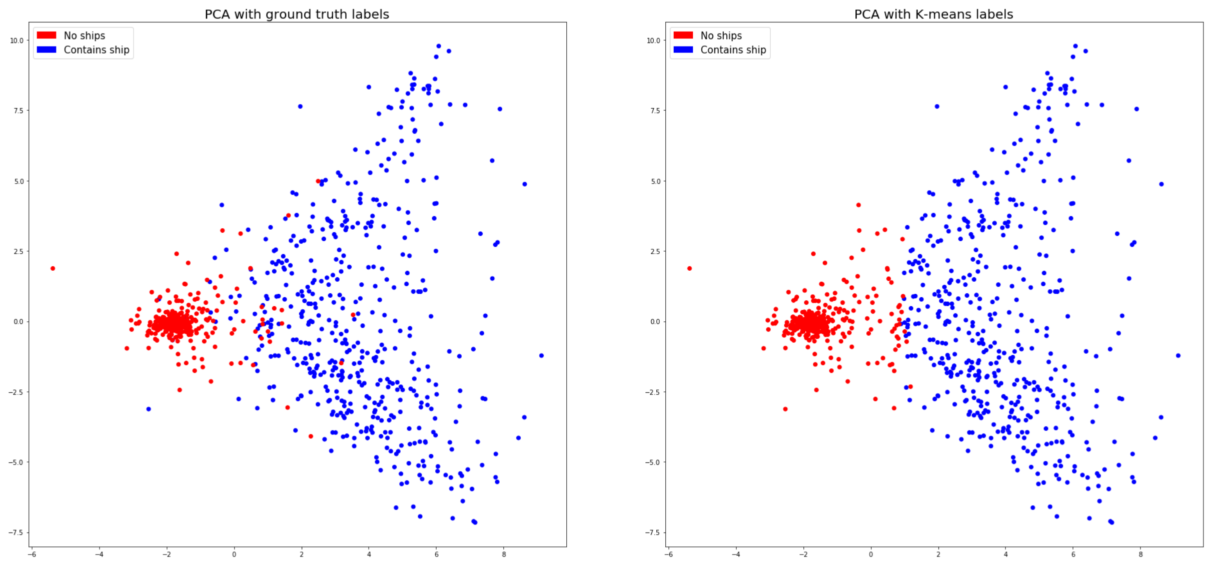
(f)



(g)

Figure 5.8: 2-component PCA for $d_z = 256$ and $\beta = 0$ (a), $\beta = 0.3$ (b), $\beta = 0.7$ (c), $\beta = 1$ (d), $\beta = 2$ (e), $\beta = 4$ (f), $\beta = 10$ (g). Each image has the ground truth labels of the test set.

In Figure 5.9 are depicted two plots of the 2-component PCA for $d_z = 256$ and $\beta = 0.3$, one with the ground truth labels (a) and another with the labels assigned by the K-means algorithm (b). It is noticeable how data points further away from the cluster of images without ships were classified as positive ("contains ship"), whereas data points closer to this cluster were classified as negative ("no ships").



(a) PCA with ground truth labels (b) PCA with K-means labels

Figure 5.9: Comparison between ground truth and K-means generated labels for $\beta = 0.3$ and $d_z = 256$.

Hereafter a deeper analysis of the model that performed better with K-means (trained with $\beta = 0.3$ and $d_z = 256$ is presented. Firstly, we should inspect the confusion matrix for the model, depicted in Figure 5.10.

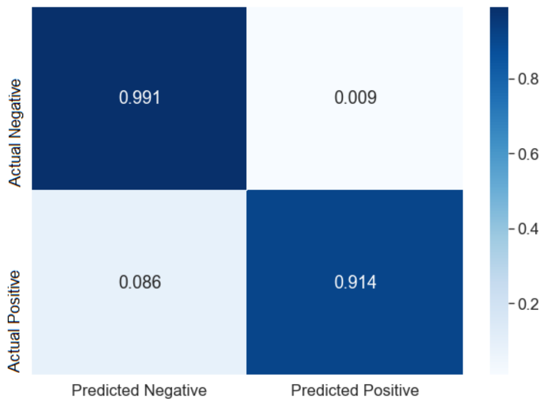


Figure 5.10: Confusion matrix for $\beta = 0.3$ and $d_z = 256$.

Based on this matrix, we conclude that roughly 99.1% of the images without ships were predicted as negative, but a smaller percentage of the images with ships were predicted as positive (about 91.4%). This means that the main source of errors of the model is in the misclassification of images with ships, missing the classification of around 8.6% of the cases, translating into 43 mislabelled anomalous images.

When analysing a machine learning model, it is of paramount importance to carefully inspect its most common mistakes, in order to infer where we should try to improve it. Additionally, it is common to find some peculiarities among the dataset, that usually justify these mistakes, such as samples with incorrect ground truth.

In fact, after taking a closer look at all the images with missed detections, 9 were found not to contain any ship at all. Investigating the source images from which these data points were extracted, we concluded that there may have occurred some mistakes in their labelling in the original data. Therefore, these images should not be accounted as missed detections, but correct negative predictions instead.

If these images are accounted as correct predictions of negative samples, the actual F1-score, accuracy, precision, and recall values for K-means applied to the mean space generated by the model trained with $\beta = 0.3$ and $d_z = 256$ are the following:

$$F_1 = 0.9551$$

$$Accuracy = 97.13\%$$

$$Precision = 0.9808$$

$$Recall = 0.9308$$

Figure 5.11 contains a few examples that were mislabelled as not containing ships by the algorithm, that actually contained ships. Inspecting these images, it is hard to identify a clear ship shape, and for that reason the model may have struggled to find enough features to trigger detection.

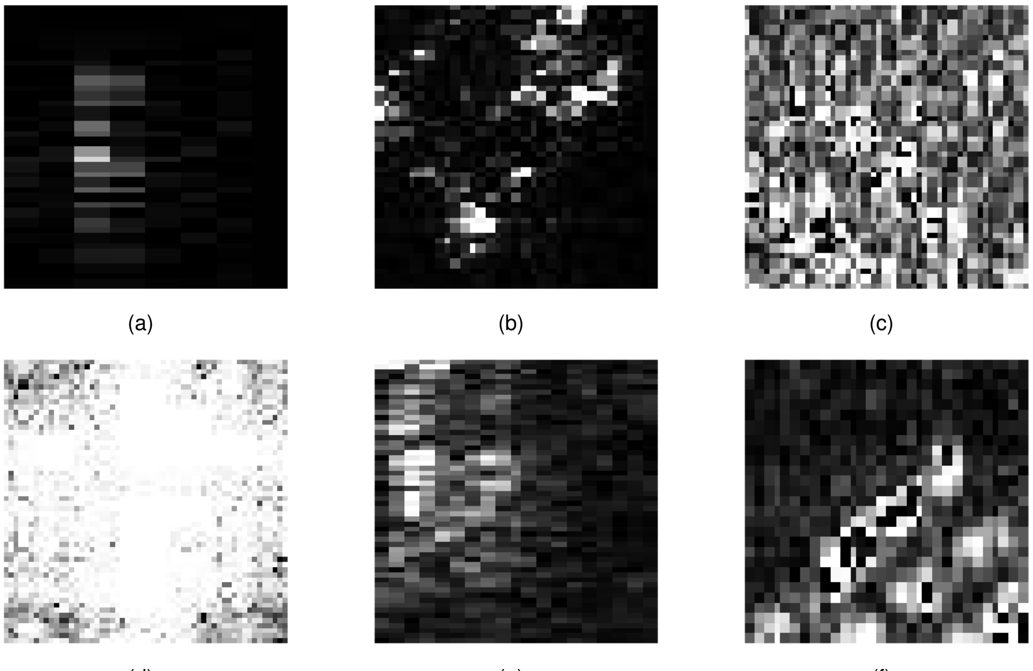


Figure 5.11: Examples of K-means missed detections for the model trained with $\beta = 0.3$ and $d_z = 256$.

Finally, we can observe the location of these images in the latent space in Figure 5.12. One should notice that the original images have different aspect ratios and sizes, and the way they are represented here is how the model receives them as inputs, resized to 56x56 pixels. Most of these missed detections lie in an area between the main cluster of normal images and the rest of the images with ships, where there is some superposition between data of the two classes. This is expected, since most of these images do not have very clear characteristics to allow the model to classify them.

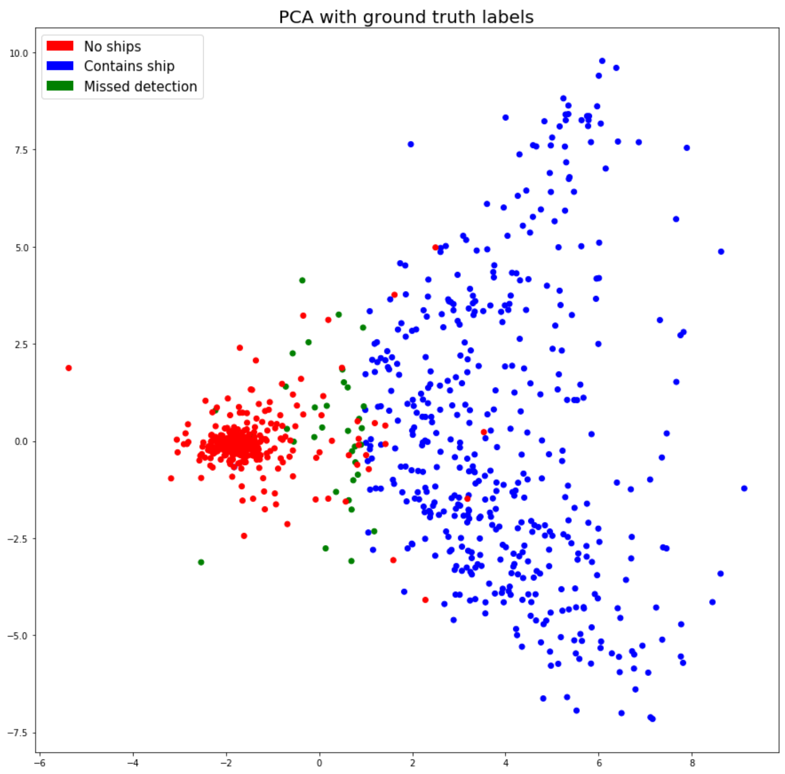


Figure 5.12: PCA with ground truth labels and missed detections, for $\beta = 0.3$ and $d_z = 256$.

Gaussian Mixture Models

Now that the analysis of the results of the model using K-means is complete, hereafter we evaluate the performance of the model using Gaussian mixture models clustering.

Before proceeding, the following notation is introduced, to facilitate the reading of this part of the section:

- The model trained with $\beta = 0.3$ and $d_z = 256$ will be called model A
- The model trained with $\beta = 1$ and $d_z = 256$ will be referred to as model B

This will be useful since these models will be mentioned multiple times throughout the rest of the section.

Table 5.5 presents the accuracy results using GMMs clustering applied to the generated latent space for each parameter pair (β, d_z). The highest accuracy for each value of β is highlighted in bold, whereas the best accuracy values for each value of latent dimensions d_z are underlined.

		Accuracy (%)				
		$d_z = 4$	$d_z = 16$	$d_z = 64$	$d_z = 128$	$d_z = 256$
	0	<u>**89.07**</u>	86.40	67.00	63.73	63.53
	0.3	77.60	68.80	80.47	86.33	<u>**96.73**</u>
	0.7	64.80	77.67	86.40	89.67	**96.53**
β	1	71.60	79.33	88.20	92.27	**96.67**
	2	74.20	84.27	92.07	93.67	92.73
	4	82.80	88.60	<u>93.73</u>	<u>**93.87**</u>	88.93
	10	87.00	<u>89.87</u>	**92.87**	91.60	85.80

Table 5.5: Accuracy of anomaly detection obtained with GMMs using generated mean space, with the highest value highlighted in bold.

Similarly to what occurred with K-means, the encoded z spaces that showed the best error rate with GMMs were the ones obtained by the models trained with $d_z = 256$ and $\beta = \{0.3, 0.7, 1\}$, with considerably higher accuracy than the others. Furthermore, for these values of β there was the same tendency of increasing the accuracy as we increased the latent space dimensions d_z. The overall top accuracy value was again obtained by applying the clustering algorithm to the mean space generated by model A, reaching the 96.73% mark, a slightly better value than the one obtained by applying K-means.

There was, however, a slight deterioration in the error rates for smaller values of d_z, specially for $d_z = 4$ and $d_z = 16$. Also, as opposed to the previous case, $\beta = 0.3$ does not yield the best accuracy values for all the latent space dimensions.

Figure 5.13 depicts the evolution of accuracy with the change of β. The tendency of decreasing the accuracy for values of β above 0.3 verified for K-means clearly does not hold for GMMs. Instead, it increases for the majority of the latent space dimensions, except for $d_z = 256$. Namely, for the models trained with latent space dimensions $d_z = \{16, 64, 128\}$, the highest accuracy values emerge when they are trained with values of β of 4 or 10. This suggests that the GMMs algorithm is probably taking better advantage of the disentanglement promoted by these values of β than the K-means algorithm. In addition, the fact that K-means assumes spherical clusters, while GMMs has more flexibility in terms of cluster shape, might justify these results. Nevertheless, these properties did not enable the algorithm to surpass the results obtained by the models trained with β values of 0.3, 0.7 and 1.

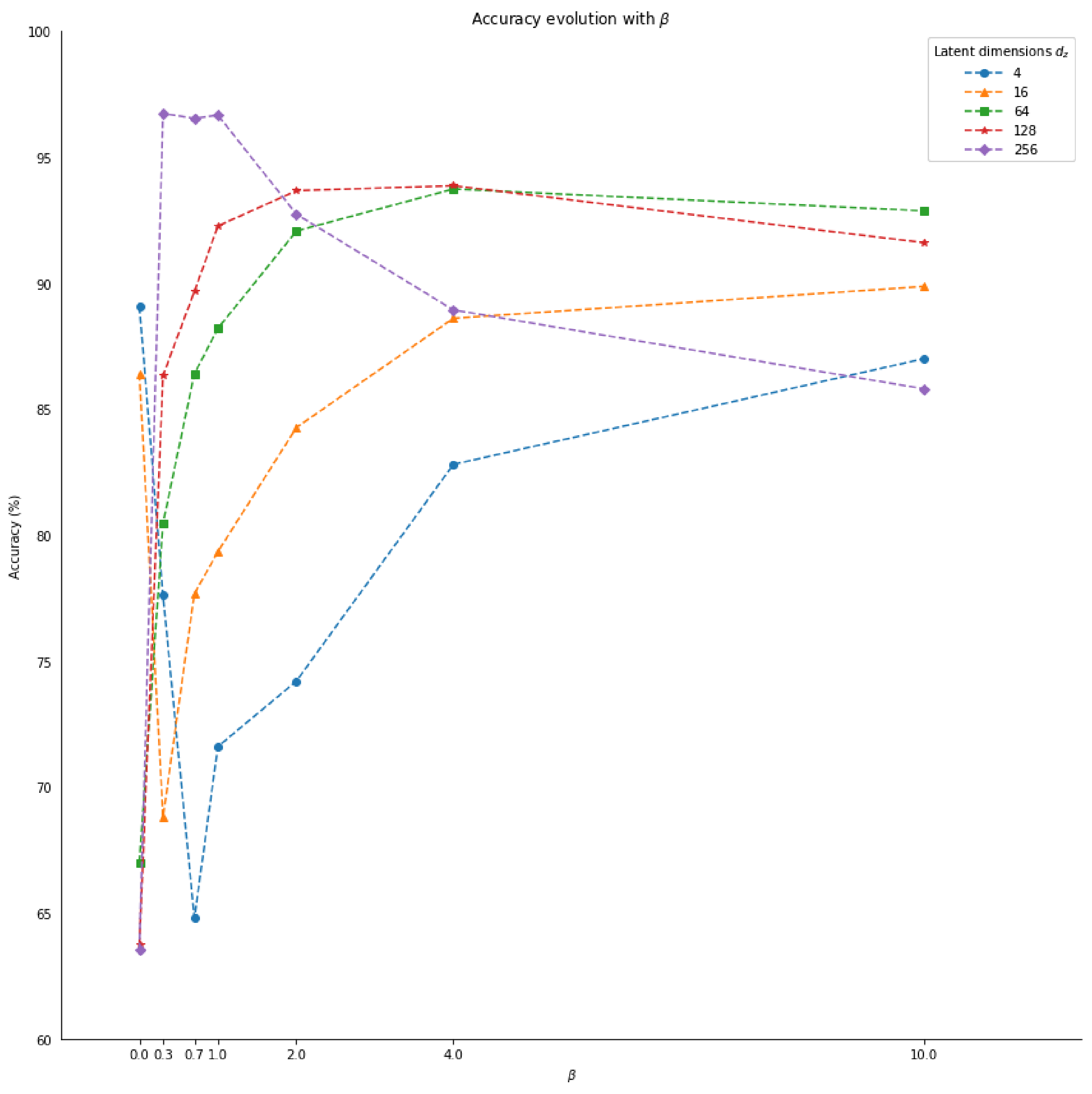


Figure 5.13: Evolution of accuracy over β.

To drive further conclusions about GMMs, we should inspect the resulting F1-scores, Precision and Recall values for this clustering algorithm, presented in Tables 5.6, 5.7, and 5.8, respectively.

For this clustering algorithm, the maximum F1-score of 0.9494 was obtained by both models A and B, a slightly better score than the best score obtained by K-means clustering. One should once more realise that the model could be trained with latent dimension values above 256, in order to see if there is a d_z value that yields better results, since the obtained results once again suggest that training with higher dimensions would possibly improve accuracy.

		F1-score				
		$d_z = 4$	$d_z = 16$	$d_z = 64$	$d_z = 128$	$d_z = 256$
	0	0.8569	0.8280	0.6685	0.6075	0.6090
	0.3	0.7462	0.6808	0.7709	0.8267	**0.9494**
	0.7	0.6540	0.7476	0.8283	0.8634	0.9467
β	1	0.7000	0.7615	0.8473	0.8942	**0.9494**
	2	0.7198	0.8066	0.8917	0.9115	0.8842
	4	0.7913	0.8517	0.9125	0.9130	0.8122
	10	0.8285	0.8657	0.8999	0.8750	0.7697

Table 5.6: F1-score of anomaly detection obtained with GMMs using generated mean space, with the highest value highlighted in bold.

		Precision				
		$d_z = 4$	$d_z = 16$	$d_z = 64$	$d_z = 128$	$d_z = 256$
	0	0.7601	0.7157	0.5025	0.4752	0.4739
	0.3	0.5995	0.5166	0.6329	0.7160	**0.9808**
	0.7	0.4864	0.5998	0.7151	0.7717	0.9706
β	1	0.5402	0.6188	0.7451	0.8221	0.9611
	2	0.5641	0.6833	0.8180	0.8534	0.9433
	4	0.6644	0.7519	0.8537	0.8656	0.9349
	10	0.7394	0.7753	0.8453	0.8681	0.8376

Table 5.7: Precision of anomaly detection obtained with GMMs using generated mean space, with the highest value highlighted in bold.

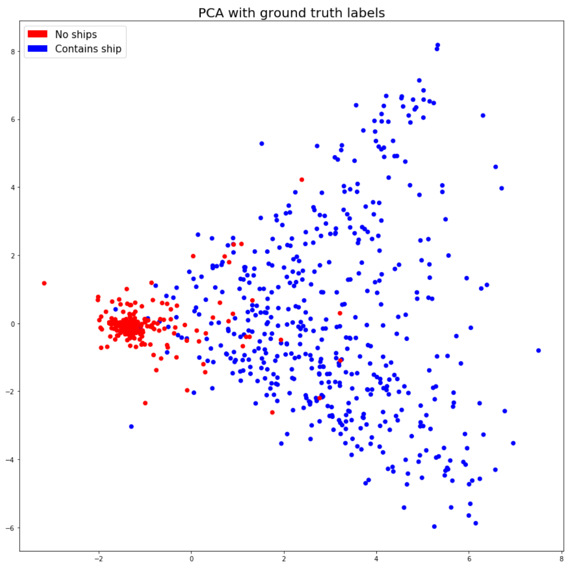
		Recall				
		$d_z = 4$	$d_z = 16$	$d_z = 64$	$d_z = 128$	$d_z = 256$
	0	0.982	0.982	**0.998**	0.842	0.852
	0.3	0.988	**0.998**	0.986	0.978	0.920
	0.7	**0.998**	0.992	0.984	0.980	0.924
β	1	0.994	0.990	0.982	0.980	0.938
	2	0.994	0.984	0.980	0.978	0.832
	4	0.978	0.982	0.980	0.966	0.718
	10	0.942	0.980	0.962	0.882	0.712

Table 5.8: Recall of anomaly detection obtained with GMMs using generated mean space, with the highest value highlighted in bold.

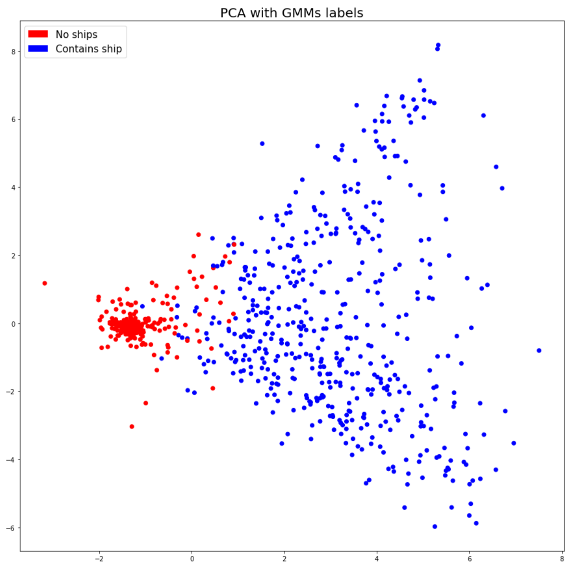
Inspecting Tables 5.7 and 5.8, it is evident that the predictions generated by GMMs generally have an overall higher recall than precision, differently to what was observed with K-means. This means that this algorithm is, in most cases, producing more false alarms than K-means.

Although model A has a slightly higher accuracy than model B, this difference is rather small to be used as a criterion of choice to elect the best between the two. Looking at precision and recall values, despite having lower precision (0.9611 as opposed to 0.9808 for model A), model B shows a considerably higher recall value (0.938 as opposed to 0.920 shown by model A), which is important if we want to minimise the number of missed detections. Since the main goal of this framework is to detect anomalies, we consider that trading off some precision (meaning it will produce more false alarms) for better anomaly detectability is very reasonable (considering the magnitude of the difference), so this is the model chosen as being the best when applying GMMs among the 35 models.

The two plots in Figure 5.14 depict the 2-component PCA applied to the mean space generated by model B, one with the ground truth labels (a) and another with the labels assigned by the GMMs algorithm (b). It is once more noticeable how data points further away from the cluster of images without ships were classified as positive ("contains ship"), whereas data points closer to this cluster were classified as negative ("no ships"). In Figure 5.9 (b) (depicting the two-component PCA representation with K-means labels) it was possible to observe that the separation made by the K-means algorithm was heavily expressed along the first dimension (x-axis in the figure) of the PCA representation. In Figure 5.14, this is not observed so evidently with the GMMs labels. Instead, it appears the model was able to make use of additional features to perform its classification, since it was capable of finding normal images that are further away from the main cluster, and ship images which are closer to it, which might justify its better performance when compared to K-means.



(a) PCA with ground truth labels



(b) PCA with GMMs labels

Figure 5.14: Comparison between ground truth and GMMs generated labels for $\beta = 1$ and $d_z = 256$.

To get a better understanding of how the model is performing on the test set, we first analyse its normalised confusion matrix, depicted in Figure 5.15.

Based on this matrix, we observe that roughly 98.1% of the images without ships were predicted as negative, but a smaller percentage of the images with ships were predicted as positive (about 93.8%). Similarly to the K-means classification results, this means that the main source of errors of the model

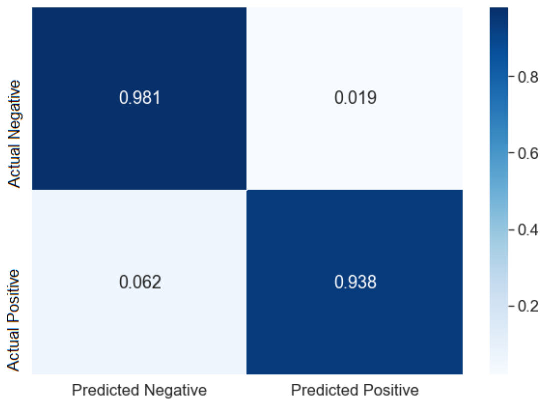


Figure 5.15: Confusion matrix for $\beta = 1$ and $d_z = 256$.

is in the misclassification of images with ships, missing the classification of around 6.2% of the cases, translating into 31 missed detections.

It is now important to perform a more in depth analysis to the misclassified images in the test set. Starting by inspecting the missed detections, it is expected that at least 9 of the 31 images with ships with wrongly predicted labels (false negatives) do not have any ship at all, corresponding to the images in the test set with incorrect ground truth positive labels, previously pointed out in the K-means analysis. After checking each of these 31 images, it was possible to identify those 9 images among the list of false negatives. Since these predictions were actually correct (the algorithm labelled them as negative, and the ground truth labels should be changed to negative), the actual total number of missed detections was 22 instead of 31.

Considering this, the new F1-score, accuracy, precision, and recall for GMMs applied to model B are the following:

$$F_1 = 0.9581$$

$$Accuracy = 97.27\%$$

$$Precision = 0.9611$$

$$Recall = 0.9552$$

Continuing this analysis, Figure 5.16 shows some examples of images with ships that were previously not detected by the K-means algorithm applied to model A's generated latent space, and were now detected by GMMs applied to model B's generated space. In these images there is a noticeable pattern: all the ships have a great amount of black pixels within their bodies, and that might have been enough to impede the K-means to recognize them.

Figure 5.17 contains 6 examples of ship images with incorrect labels assigned by GMMs applied to the mean space generated by model B. The model might have failed to identify these shapes due to the dimness of the pixels that compose the ships, probably mistaking them with the kind of noise that is typically present in most SAR images.

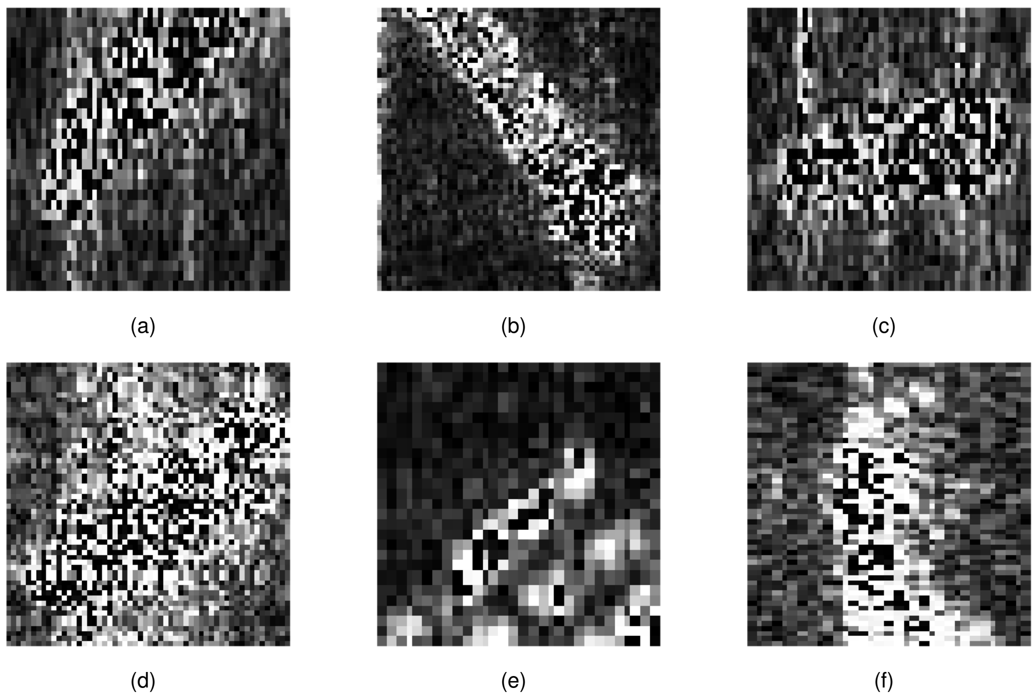


Figure 5.16: Examples of images with ships not detected by K-means and detected by GMMs.

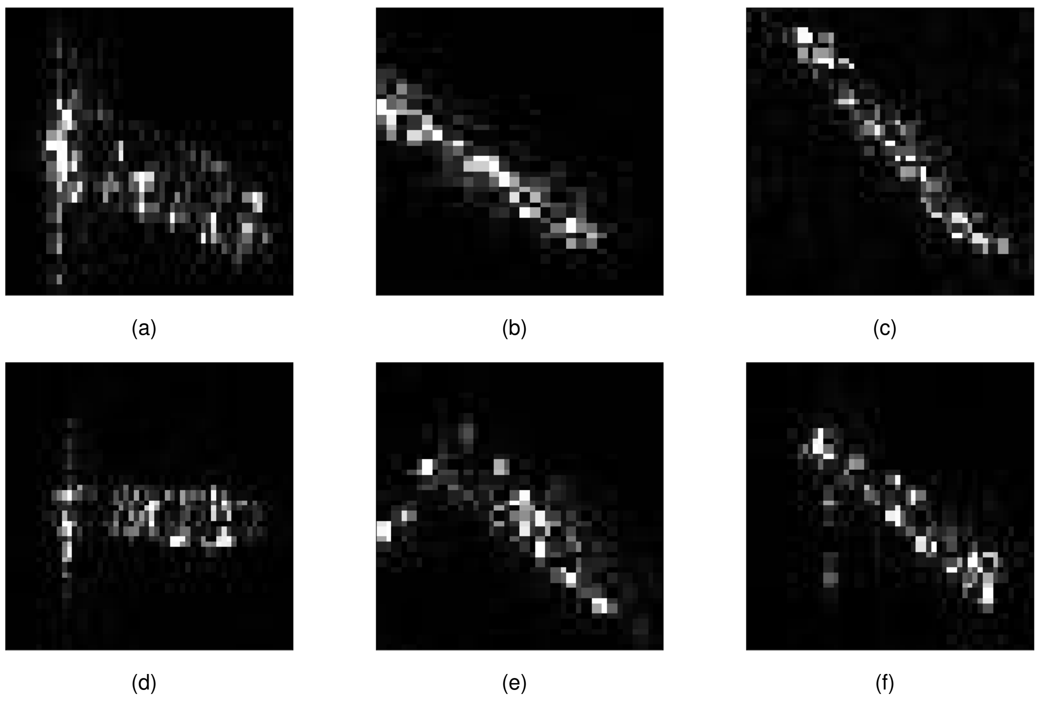


Figure 5.17: Examples of GMMs missed detections for the model trained with $\beta = 1$ and $d_z = 256$.

5.2.2 Comparison with a supervised CNN

The normalised confusion matrix for the classification results of the supervised CNN is depicted in Figure 5.18.

Comparing the values of this matrix with the values of the confusion matrices for the classification with our proposed method, the major difference between them is in the ship detection rate, that is

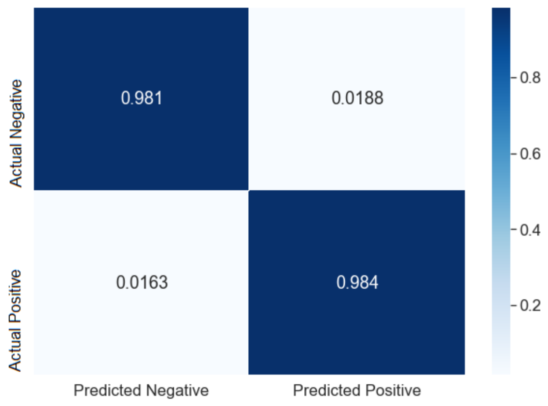


Figure 5.18: Confusion matrix for the supervised CNN.

considerably higher with the supervised CNN.

The overall results of the proposed framework with both clustering algorithms and the results of the supervised CNN are shown in Table 5.9. Notice that all the presented values for each evaluation metric are taking into account the 9 images that had incorrect ground truth labels, and were correctly labelled by the algorithms.

Evaluation metric Used	**Clustering Algorithm Used**		**Supervised CNN**
	K-means	***GMMs***	
Accuracy	0.9713	0.9727	0.9820
F1-score	0.9551	0.9581	0.9728
Precision	0.9808	0.9611	0.9622
Recall	0.9308	0.9552	0.9837

Table 5.9: Results overview for all the used algorithms.

Regarding the two clustering methods, the Gaussian Mixture Models clustering shows slightly better accuracy (+0.14%) and F1-score (+0.31%), as well as considerably higher recall (+2.62%), when compared to the K-means algorithm. K-means, on the other hand, presents a considerably higher precision (+2.05%), when compared to the Gaussian Mixture Models.

As mentioned before, when the main goal is to detect positives and, consequently, avoid false negatives, it is important to choose the method that reveals a higher recall value. Therefore, and even though having lower precision, the algorithm that performed the classification task better was the Gaussian Mixture Models clustering.

Finally, it is important to compare the performance of the proposed framework using GMMs with the performance of the supervised CNN. Inspecting table 5.9, it is evident that the latter performed better in every evaluation metric used, especially in terms of recall, meaning it had significantly more success in detecting images with ships. These results are well aligned with what was expected, because a supervised model has clear advantages when it comes to learning. Namely, since they are provided with images containing ships during training - which is not the case for the proposed unsupervised method, that is trained only with normal images - they are able to learn from them in order to distinguish between both types of images at test time, justifying the much better recall values.

Chapter 6

Conclusions

The presented thesis aimed at developing a completely unsupervised machine learning framework to detect anomalies in SAR images, resorting to the representation learning ability of variational autoencoders. By the time when we engaged in this endeavour, we believed that unsupervised learning tools were especially important in the field of remote sensing, namely in the field of SAR image analysis, given the huge and ever-increasing amount of data available. This work came to confirm our belief.

To design and test the proposed framework, images of the ocean were chosen due to the relevance of monitoring areas that are mostly remote. In addition, given the vastness of the ocean, the great majority of sea images is expected to not contain any anomaly at all, therefore being great candidates for a successful model. Nevertheless, we believe that other cases where the observed terrain does not change much over time could also take advantage of this solution, provided that enough data is available. In these cases, the network could also have to be adapted to the complexity of the images involved, since more analytical power could be necessary for such scenarios.

Since very few previous works were found on the application of VAEs to SAR images, arriving at the final model was no easy task. A lot of experimenting was conducted in order to achieve satisfying results. This process, as is usual when training a neural network for a new application, was limited by the computational resources available.

Nonetheless, the final model achieved very interesting results, being able to classify the test set images with 97.27% of accuracy and an F1-score of 0.9581, so it is fair to conclude that the proposed objective for this thesis was reached with success.

6.1 Achievements

The main goal of this work was to provide an unsupervised way of detecting anomalies in SAR images, using VAEs as the main stepping stone. Given that the proposed framework was able to detect roughly 95.5% of the anomalous images in the test set, with a false alarm rate of 0.019 and a missed detection rate of 0.045, we consider that this goal was achieved.

A paper related to the work presented here was submitted and accepted for publication and oral presentation at the 2020 IEEE International Geoscience and Remote Sensing Symposium (IGARSS2020).

6.2 Future Work

Despite the good results obtained, there are a few things that should be considered for possible future developments.

As previously mentioned, the network should be trained with $d_z > 256$, in order to check for possible improvements.

Regarding the applicability of the solution to real case scenarios, further testing should be conducted to evaluate other performance metrics, such as the time needed to classify each image, in order to perform a benchmark with current state of the art methods, and assess the feasibility of applying the framework in a live surveillance system. This would also require a study of possible optimisations to be applied to the classification pipeline, in order to achieve minimum classification time.

This work focused on analysing the generated mean spaces of each trained model. Future developments could try to also take advantage of the generated sets of variances, that could possibly contain useful information to help the model perform even better. Further studies could also include training the model using the triplet loss [136], and the usage of adversarial VAEs [137].

Finally, it would be interesting to train and test the network in other scenarios, such as in land surveillance applications, in order to analyse how the model would behave in those conditions.

Bibliography

[1] UNHCR. *Desperate Journeys*. Online, 2019. Available: https://www.unhcr.org/desperatejourneys/.

[2] A. Baker. *A week on board a refugee recovery ship*. Time, 2019. Available: https://time.com/refugee-rescue/.

[3] FAO. *The State of World Fisheries and Aquaculture 2018 - Meeting the sustainable development goals*. 2018. Rome.

[4] D. J. Agnew, J. Pearce, G. Pramod, T. Peatman, R. Watson, J. R. Beddington, and T. J. Pitcher. Estimating the worldwide extent of illegal fishing. *PLoS ONE*, 4(2):e4570, 2009.

[5] FAO. *FAO Workshop on Vulnerable Ecosystems and Destructive Fishing in Deep-sea Fisheries.* 2007. Rome.

[6] G. Cheng, J. Han, and X. Lu. Remote sensing image scene classification: Benchmark and state of the art. *Proceedings of the IEEE*, 105(10):1865–1883, 2017.

[7] D. Crisp. The state-of-the-art in ship detection in synthetic aperture radar imagery. *DSTO Information Science Laboratory*, 2004.

[8] P. Iervolino and R. Guida. A novel ship detector based on the generalized-likelihood ratio test for SAR imagery. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 10(8):3616–3630, 2017.

[9] X. Leng, K. Ji, K. Yang, and H. Zou. A bilateral CFAR algorithm for ship detection in SAR images. *IEEE Geoscience and Remote Sensing Letters*, 12(7):1536–1540, 2015.

[10] A. Marino, W. Dierking, and C. Wesche. A depolarization ratio anomaly detector to identify icebergs in sea ice using dual-polarization SAR images. *IEEE Transactions on Geoscience and Remote Sensing*, 54(9):5602–5615, 2016.

[11] A. Marino and P. Iervolino. Ship detection with Cosmo-SkyMed PINGPONG data using the dual-pol ratio anomaly detector. In *2017 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, pages 3897–3900, 2017.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[13] H. Khan and C. Yunze. Ship detection in SAR image using YOLOv2. In *Proceedings of the 37th Chinese Control Conference*, pages 9495–9499, 2018.

[14] Y.-L. Chang, A. Anagaw, L. Chang, Y. C. Wang, C.-Y. Hsiao, and W.-H. Lee. Ship detection based on YOLOv2 for SAR imagery. *Remote Sensing*, 11(7):786, 2019.

[15] J. Li, C. Qu, and J. Sun. Ship detection in SAR images based on an improved faster R-CNN. In *SAR in Big Data Era: Models, Methods & Applications*, pages 1–6, 2017.

[16] M. Kang, X. Leng, Z. Lin, and K. Ji. A modified faster R-CNN based on CFAR algorithm for SAR ship detection. In *International Workshop on Remote Sensing with Intelligent Processing*, pages 1–4, 2017.

[17] J. Zhao, Z. Zhang, W. Yu, and T. Truong. A cascade coupled convolutional neural network guided visual attention method for ship detection from SAR images. *IEEE Access*, 6:50693–50708, 2018.

[18] Y. Li, Z. Ding, C. Zhang, Y. Wang, and J. Chen. SAR ship detection based on Resnet and transfer learning. In *IGARSS 2019 - 2019 IEEE International Geoscience and Remote Sensing Symposium*, pages 1188–1191, 2019.

[19] X. X. Zhu, D. Tuia, L. Mou, G. Xia, L. Zhang, F. Xu, and F. Fraundorfer. Deep learning in remote sensing: A comprehensive review and list of resources. *IEEE Geoscience and Remote Sensing Magazine*, 5(4):8–36, 2017.

[20] D. Crisp. The state-of-the-art in ship detection in synthetic aperture radar imagery. 2004.

[21] C. C. Wackerman, K. S. Friedman, W. G. Pichel, P. Clemente-Colón, and X. Li. Automatic detection of ships in RADARSAT-1 SAR imagery. *Canadian Journal of Remote Sensing*, 27(5):568–577, 2001.

[22] K. Eldhuset. Automatic ship and ship wake detection in spaceborne SAR images from coastal regions. In *International Geoscience and Remote Sensing Symposium, 'Remote Sensing: Moving Toward the 21st Century'.*, volume 3, pages 1529–1533, 1988.

[23] K. Eldhuset. An automatic ship and ship wake detection system for spaceborne SAR images in coastal regions. *IEEE Transactions on Geoscience and Remote Sensing*, 34(4):1010–1019, 1996.

[24] T. Wahl, K. Eldhuset, and Skøelv. Ship traffic monitoring using the ERS-1 SAR. In *First ERS-1 Symposium - Space at the Service of our Environment*, volume 359, pages 823–823, 1992.

[25] M. N. Ferrara and A. Torre. Alenia Aerospazio activity on SAR data analysis and information extraction. In *Image Processing, Signal Processing, and Synthetic Aperture Radar for Remote Sensing*, volume 3217, pages 67–75, 1997.

[26] Jong-sen Lee and I. Jurkevich. Coastline detection and tracing in SAR images. *IEEE Transactions on Geoscience and Remote Sensing*, 28(4):662–668, 1990.

[27] X. Descombes, M. Moctezuma, H. Maître, and J.-P. Rudant. Coastline detection by a markovian segmentation on SAR images. *Signal Process.*, 55(1):123–132, 1996.

[28] X. Ding and X. Li. Coastline detection in SAR images using multiscale normalized cut segmentation. In *2014 IEEE Geoscience and Remote Sensing Symposium*, pages 4447–4449, 2014.

[29] F. Nunziata, M. Migliaccio, and X. Li. Dual-polarized COSMO-SkyMed SAR data for coastline detection. In *2012 IEEE International Geoscience and Remote Sensing Symposium*, pages 5109–5112, 2012.

[30] G. Sheng, W. Yang, X. Deng, C. He, Y. Cao, and H. Sun. Coastline detection in synthetic aperture radar (SAR) images by integrating watershed transformation and controllable gradient vector flow (GVF) snake model. *IEEE Journal of Oceanic Engineering*, 37(3):375–383, 2012.

[31] M. Schmitt, L. Wei, and X. X. Zhu. Automatic coastline detection in non-locally filtered tandem-x data. In *2015 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, pages 1036–1039, 2015.

[32] F. Argenti, G. Benelli, A. Garzelli, and A. Mecocci. Automatic ship detection in SAR images. In *92 International Conference on Radar*, pages 465–468, 1992.

[33] G. Benelli, A. Garzelli, and A. Mecocci. Complete processing system that uses fuzzy logic for ship detection in SAR images. *IEE Proceedings - Radar, Sonar and Navigation*, 141(4):181–186, 1994.

[34] P. Wang, J. Chong, and H. Wang. Ship detection of the airborne SAR images. In *IGARSS 2000. IEEE 2000 International Geoscience and Remote Sensing Symposium. Taking the Pulse of the Planet: The Role of Remote Sensing in Managing the Environment. Proceedings (Cat. No.00CH37120)*, volume 1, pages 348–350, 2000.

[35] D. Devapal, N. Hashna, V. Aparna, C. Bhavyasree, J. Mathai, and K. S. Soman. Object detection from SAR images based on curvelet despeckling. *Materials Today: Proceedings*, 11:1102–1116, 2019.

[36] A. J. Rye, F. G. Sawyer, and R. Sothinathan. A workstation for the fast detection of ships. In *10th Annual International Symposium on Geoscience and Remote Sensing*, pages 2263–2266, 1990.

[37] M. N. Ferrara, A. Gallon, and A. Torre. Improvement in automatic detection and recognition of moving targets in Alenia Aerospazio activity. In *Image and Signal Processing for Remote Sensing IV*, volume 3500, pages 96–103, 1998.

[38] I.-I. Lin and V. Khoo. Computer-based algorithm for ship detection from ERS SAR imagery. 1997.

[39] I-I Lin, Leong Keong Kwoh, Yuan-Chung Lin, and V. Khoo. Ship and ship wake detection in the ERS SAR imagery using computer-based algorithm. In *IGARSS'97. 1997 IEEE International Geoscience and Remote Sensing Symposium Proceedings. Remote Sensing - A Scientific Vision for Sustainable Development*, volume 1, pages 151–153, 1997.

[40] K. El-Darymli, P. Mcguire, D. Power, and C. Moloney. Target detection in synthetic aperture radar imagery: a state-of-the-art survey. *Journal of Applied Remote Sensing*, 7(1):071598, 2013.

[41] J. C. Principe, A. Radisavljevic, J. Fisher, M. Hiett, and L. M. Novak. Target prescreening based on a quadratic gamma discriminator. *IEEE Transactions on Aerospace and Electronic Systems*, 34(3):706–715, 1998.

[42] M. di Bisceglie and C. Galdi. CFAR detection of extended objects in high-resolution SAR images. *IEEE Transactions on Geoscience and Remote Sensing*, 43(4):833–843, 2005.

[43] S. Kuttikkad and R. Chellappa. Non-gaussian CFAR techniques for target detection in high resolution SAR images. In *Proceedings of 1st International Conference on Image Processing*, volume 1, pages 910–914, 1994.

[44] M. Liao, C. Wang, Y. Wang, and L. Jiang. Using SAR images to detect ships from sea clutter. *IEEE Geoscience and Remote Sensing Letters*, 5(2):194–198, 2008.

[45] J. Xu, Wei Han, X. He, and Ren-xi Chen. Small target detection in SAR image using the alpha-stable distribution model. In *2010 International Conference on Image Analysis and Signal Processing*, pages 64–68, 2010.

[46] J. Salazar. *Detection schemes for synthetic aperture radar imagery based on a beta prime statistical model.* PhD thesis, University of New Mexico, 1999.

[47] H. M. Finn. Adaptive detection mode with threshold control as a function of spatially sampled clutter level estimates. volume 29, pages 414–465, 1968.

[48] H. Rohling. Radar CFAR thresholding in clutter and multiple target situations. *IEEE Transactions on Aerospace and Electronic Systems*, (4):608–621, 1983.

[49] M. Rimbert and M. Bell. Multiresolution order-statistic CFAR techniques for radar target detection. *Proceedings of SPIE - The International Society for Optical Engineering*, 5674:282–292, 2005.

[50] P. P. Gandhi and S. A. Kassam. Analysis of CFAR processors in nonhomogeneous background. *IEEE Transactions on Aerospace and Electronic Systems*, 24(4):427–445, 1988.

[51] L. M. Novak and S. R. Hesse. On the performance of order-statistics CFAR detectors. In *[1991] Conference Record of the Twenty-Fifth Asilomar Conference on Signals, Systems Computers*, pages 835–836, 1991.

[52] A. Tom and R. Viswanathan. Switched order statistics CFAR test for target detection. In *2008 IEEE Radar Conference*, pages 1–5, 2008.

[53] T. . Van Cao. A CFAR thresholding approach based on test cell statistics. In *Proceedings of the 2004 IEEE Radar Conference (IEEE Cat. No.04CH37509)*, pages 349–354, 2004.

[54] G. Gao. A parzen-window-kernel-based CFAR algorithm for ship detection in SAR images. *IEEE Geoscience and Remote Sensing Letters*, 8(3):557–561, 2011.

[55] S. R. Tian, C. Wang, and H. Zhang. An improved nonparametric CFAR method for ship detection in single polarization synthetic aperture radar imagery. In *2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, pages 6637–6640, 2016.

[56] Y. Cui, J. Yang, Y. Yamaguchi, G. Singh, S. Park, and H. Kobayashi. On semiparametric clutter estimation for ship detection in synthetic aperture radar images. *IEEE Transactions on Geoscience and Remote Sensing*, 51(5):3170–3180, 2013.

[57] K. Ouchi, S. Tamaki, H. Yaguchi, and M. Iehara. Ship detection based on coherence images derived from cross correlation of multilook SAR images. *IEEE Geoscience and Remote Sensing Letters*, 1(3):184–187, 2004.

[58] D. Howard, S. Roberts, and R. Brankin. Target detection in SAR imagery by genetic programming. In *Advances in Engineering Software*, volume 30, pages 303–311, 1999.

[59] A. Marino and I. Hajnsek. Statistical tests for a ship detector based on the polarimetric notch filter. *IEEE Transactions on Geoscience and Remote Sensing*, 53(8):4578–4595, 2015.

[60] A. Marino, N. Walker, and I. H. Woodhouse. Ship detection using SAR polarimetry. the development of a new algorithm designed to exploit new satellite SAR capabilities for maritime surveillance. In *Proceedings of the Third International Workshop SeaSAR 2010*, 2010.

[61] L. Zhai, Y. Li, and Y. Su. Inshore ship detection via saliency and context information in high-resolution SAR images. *IEEE Geoscience and Remote Sensing Letters*, 13(12):1870–1874, 2016.

[62] E. Conte, M. Lops, and G. Ricci. Radar detection in K-distributed clutter. *IEE Proceedings - Radar, Sonar and Navigation*, 141(2):116–118, 1994.

[63] P. Iervolino, R. Guida, and P. Whittaker. A novel ship-detection technique for Sentinel-1 SAR data. In *2015 IEEE 5th Asia-Pacific Conference on Synthetic Aperture Radar (APSAR)*, pages 797–801, 2015.

[64] S. Chen and H. Wang. SAR target recognition based on deep learning. In *2014 International Conference on Data Science and Advanced Analytics (DSAA)*, pages 541–547, 2014.

[65] E. R. Keydel, S. W. Lee, and J. T. Moore. MSTAR extended operating conditions: a tutorial. In *Algorithms for Synthetic Aperture Radar Imagery III*, volume 2757, pages 228–242, 1996.

[66] D. A. Morgan. Deep convolutional neural networks for ATR from SAR imagery. In *Algorithms for Synthetic Aperture Radar Imagery XXII*, volume 9475, page 94750F. International Society for Optics and Photonics, 2015.

[67] J. Ding, B. Chen, H. Liu, and M. Huang. Convolutional neural network with data augmentation for SAR target recognition. *IEEE Geoscience and remote sensing letters*, 13(3):364–368, 2016.

[68] K. Du, Y. Deng, R. Wang, T. Zhao, and N. Li. SAR ATR based on displacement-and rotation-insensitive cnn. *Remote Sensing Letters*, 7(9):895–904, 2016.

[69] M. Wilmanski, C. Kreucher, and J. Lauer. Modern approaches in deep learning for SAR ATR. In *Algorithms for synthetic aperture radar imagery XXIII*, volume 9843, page 98430N. International Society for Optics and Photonics, 2016.

[70] C. Bentes, A. Frost, D. Velotto, and B. Tings. Ship-iceberg discrimination with convolutional neural networks in high resolution SAR images. In *Proceedings of EUSAR 2016: 11th European Conference on Synthetic Aperture Radar*, pages 1–4, 2016.

[71] C. P. Schwegmann, W. Kleynhans, B. P. Salmon, L. W. Mdakane, and R. G. V. Meyer. Very deep learning for ship discrimination in synthetic aperture radar imagery. In *2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, pages 104–107, 2016.

[72] R. K. Srivastava, K. Greff, and J. Schmidhuber. Training very deep networks, 2015.

[73] M. Ma, J. Chen, W. Liu, and W. Yang. Ship classification and detection based on CNN using GF-3 SAR images. *Remote Sensing*, 10:2043, 12 2018. doi: 10.3390/rs10122043.

[74] C. Bentes, D. Velotto, and B. Tings. Ship classification in TerraSAR-X images with convolutional neural networks. *IEEE Journal of Oceanic Engineering*, 43(1):258–266, 2018.

[75] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[76] R. Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[77] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[78] M. Kang, K. Ji, X. Leng, and Z. Lin. Contextual region-based convolutional neural network with multilayer fusion for SAR ship detection. 9(8):860, 2017.

[79] J. Jiao, Y. Zhang, H. Sun, X. Yang, X. Gao, W. Hong, K. Fu, and X. Sun. A densely connected end-to-end neural network for multiscale and multiscene SAR ship detection. *IEEE Access*, 6: 20881–20892, 2018.

[80] J. Li, C. Qu, and J. Shao. Ship detection in SAR images based on an improved faster R-CNN. In *2017 SAR in Big Data Era: Models, Methods and Applications (BIGSARDATA)*, pages 1–6, 2017.

[81] Z. Lin, K. Ji, X. Leng, and G. Kuang. Squeeze and excitation rank faster r-cnn for ship detection in SAR images. *IEEE Geoscience and Remote Sensing Letters*, 16(5):751–755, 2019.

[82] J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu. Squeeze-and-excitation networks, 2017.

[83] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. 2014.

[84] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016.

[85] J. Redmon and A. Farhadi. YOLO9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[86] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015.

[87] J. Redmon and A. Farhadi. YOLOv3: An incremental improvement. 2018.

[88] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. SSD: Single shot multibox detector. *Lecture Notes in Computer Science*, pages 21–37, 2016.

[89] J. Wang, C. Lu, and W. Jiang. Simultaneous ship detection and orientation estimation in SAR images based on attention module and angle regression. *Sensors*, 18(9):2851, 2018.

[90] Y. Wang, C. Wang, H. Zhang, C. Zhang, and Q. Fu. Combing single shot multibox detector with transfer learning for ship detection using chinese gaofen-3 images. In *2017 Progress in Electromagnetics Research Symposium - Fall (PIERS - FALL)*, pages 712–716, 2017.

[91] C. Chen, C. He, C. Hu, H. Pei, and L. Jiao. A deep neural network based on an attention mechanism for SAR ship detection in multiscale and complex scenarios. *IEEE Access*, 7:104848–104863, 2019.

[92] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi. Inception-v4, Inception-ResNet and the impact of residual connections on learning, 2016.

[93] N. Bodla, B. Singh, R. Chellappa, and L. S. Davis. Soft-NMS – improving object detection with one line of code, 2017.

[94] T. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 2980–2988, 2018.

[95] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[96] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[97] Y. Wang, C. Wang, H. Zhang, Y. Dong, and S. Wei. Automatic ship detection based on retinanet using multi-resolution gaofen-3 imagery. *Remote Sensing*, 11(5):531, 2019.

[98] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[99] M. Kang, K. Ji, X. Leng, X. Xing, and H. Zou. Synthetic aperture radar target recognition with feature fusion based on a stacked autoencoder. *Sensors*, 17(1):192, 2017.

[100] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[101] L. Wolf, T. Hassner, and Y. Taigman. Descriptor based methods in the wild. In *Workshop on Faces in 'Real-Life' Images: Detection, Alignment, and Recognition*, 2008.

[102] J. Geng, J. Fan, H. Wang, X. Ma, B. Li, and F. Chen. High-resolution SAR image classification via deep convolutional autoencoders. *IEEE Geoscience and Remote Sensing Letters*, 12(11): 2351–2355, 2015.

[103] J. Geng, H. Wang, J. Fan, and X. Ma. Deep supervised and contractive neural network for SAR image classification. *IEEE Transactions on Geoscience and Remote Sensing*, 55(4):2442–2459, 2017.

[104] B. Hou, H. Kou, and L. Jiao. Classification of polarimetric SAR images using multilayer autoencoders and superpixels. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 9(7):3072–3081, 2016.

[105] L. Zhang, W. Ma, and D. Zhang. Stacked sparse autoencoder in PolSAR data classification using local spatial information. *IEEE Geoscience and Remote Sensing Letters*, 13(9):1359–1363, 2016.

[106] X. Li, C. Li, P. Wang, Z. Men, and H. Xu. SAR ATR based on dividing CNN into CAE and SNN. In *2015 Ieee 5th Asia-Pacific Conference on Synthetic Aperture Radar (Apsar)*, pages 676–679, 2015.

[107] P. Zhang, M. Gong, L. Su, J. Liu, and Z. Li. Change detection based on deep feature representation and mapping transformation for multi-spatial-resolution remote sensing images. *ISPRS Journal of Photogrammetry and Remote Sensing*, 116:24–41, 2016.

[108] M. Gong, H. Yang, and P. Zhang. Feature learning and change feature classification based on deep learning for ternary change detection in SAR images. *ISPRS Journal of Photogrammetry and Remote Sensing*, 129:212–225, 2017.

[109] S. Deng, L. Du, C. Li, J. Ding, and H. Liu. SAR automatic target recognition based on euclidean distance restricted autoencoder. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 10(7):3323–3333, 2017.

[110] W. Xie, L. Jiao, B. Hou, W. Ma, J. Zhao, S. Zhang, and F. Liu. POLSAR image classification via Wishart-AE model or Wishart-CAE model. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 10(8):3604–3615, 2017.

[111] J.-S. Lee, M. R. Grunes, and R. Kwok. Classification of multi-look polarimetric SAR imagery based on complex Wishart distribution. *International Journal of Remote Sensing*, 15(11):2299–2311, 1994.

[112] J. Geng, H. Wang, J. Fan, and X. Ma. SAR image classification via deep recurrent encoding neural networks. *IEEE Transactions on Geoscience and Remote Sensing*, 56(4):2255–2269, 2017.

[113] S. Sinha, S. Giffard-Roisin, F. Karbou, M. Deschatres, A. Karas, N. Eckert, and C. Monteleoni. Detecting avalanche deposits using variational autoencoder on sentinel-1 satellite imagery. In *NeurIPS 2019 Workshop : Tackling Climate Change with Machine Learning NeurIPS workshop*, 2019.

[114] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[115] Y. Xu, G. Zhang, K. Wang, and H. Leung. SAR target recognition based on variational autoencoder. In *2019 IEEE MTT-S International Microwave Biomedical Conference (IMBioC)*, volume 1, pages 1–4. IEEE, 2019.

[116] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[117] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th International Conference on Computer Vision*, pages 2146–2153, 2009.

[118] V. Nair and G. E. Hinton. Rectified linear units improve restricted Boltzmann machines. In *ICML*, 2010.

[119] CS231n. *Convolutional Neural Networks for Visual Recognition*. Online, Stanford University, 2020. Available: `http://cs231n.stanford.edu/`.

[120] D. R. Wilson and T. R. Martinez. The general inefficiency of batch training for gradient descent learning. *Neural networks : the official journal of the International Neural Network Society*, 16(10): 1429–1451, 2003.

[121] B. Polyak. Some methods of speeding up the convergence of iteration methods. *Ussr Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

[122] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[123] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(7), 2011.

[124] G. Hinton. Neural networks for machine learning. *Coursera, video lectures*, 2012.

[125] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2014.

[126] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1): 1929–1958, 2014.

[127] D. P. Kingma and M. Welling. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 2019.

[128] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. 2016.

[129] C. M. Bishop. *Pattern recognition and machine learning*. springer, 2006.

[130] S. Lloyd. Least squares quantization in PCM. *IEEE transactions on information theory*, 28(2): 129–137, 1982.

[131] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

[132] Y. Wang, C. Wang, H. Zhang, Y. Dong, and S. Wei. A SAR dataset of ship detection for deep learning under complex backgrounds. *Remote Sensing*, 11(7):765, 2019.

[133] Tensorflow. *Convolutional Variational Autoencoder*. Online, 2020. Available: `https://www.tensorflow.org/tutorials/generative/cvae`.

[134] F. Chollet et al. Keras. `https://keras.io`, 2015.

[135] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

[136] K. Q. Weinberger and L. K. Saul. Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research*, 10(2), 2009.

[137] L. M. Mescheder, S. Nowozin, and A. Geiger. Adversarial variational bayes: Unifying variational autoencoders and generative adversarial networks. *CoRR*, 2017.