NTNU
Norwegian University of
Science and Technology

Unsupervised Object Detection in Images from Maritime Environments

Kristoffer Kleven Krossholm

Master of Science in Cybernetics and Robotics
Submission date: June 2018
Supervisor: Edmund Førland Brekke, ITK
Co-supervisor: Arild Nøkland, Kongsberg Seatex

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Problem description

The goal of this thesis is to explore unsupervised machine learning methods from the field of computer vision to extract information from images about objects that might interfere with the operation of an autonomous vessel. The work should extend the developments of the candidate's fifth year specialization project by mitigating identified drawbacks. A promising method should be implemented and tested.

Preface

This thesis concludes my master's degree in Cybernetics and Robotics at the Norwegian University of Science and Technology.

I would like to thank my supervisors, Associate Professor Edmund Brekke Førland and Arild Nøkland at Kongsberg Seatex, for their valuable guidance and feedback throughout the work of this thesis. Also, the people at office G234 deserves gratitude for their contribution to an extraordinary academic atmosphere.

The work in this thesis has been carried out with support by Kongsberg Seatex. Kongsberg Seatex has provided image data and hardware used for implementation and computation. In addition, they have provided a simple software library for loading images and metadata into memory. Apart from this, the object detection algorithm is fully implemented by myself.

Trondheim, June 2018

Kristoffer Kleven Krossholm

Abstract

Autonomous surface vehicles (ASVs) are likely to revolutionize the maritime industry in the near future. To obtain situation awareness and avoid collisions, information from various types of sensors is needed. Visual cameras mounted on an ASV provide a detailed description of its surroundings.

In this thesis, an algorithm for object detection in images based on unsupervised learning is implemented and tested. Different from supervised learning, unsupervised learning algorithms have no ground truth to use as guidance for learning what type of objects it should detect. This saves a large amount of human labor, and enables detection of objects the algorithm has not seen before.

The object detection algorithm is based on a neural network, specifically an autoencoder, that is trained to represent visual ocean features well, while highlighting what it consider anomalous. Hence, the algorithm relies on the assumption that objects are represented as anomalies in the image data.

The evaluation shows that the algorithm at its best is able to detect all objects. However, this comes at the expense of a high number of false positives. The maritime environment contains a lot of natural features, such as waves, that stand out visually. These are often considered anomalous by the algorithm, and therefore detected as objects.

Sammendrag

Autonome overflatefartøyer vil trolig revolusjonere den maritime industrien i nær fremtid. Ulike typer sensorer er nødvendig for å gi informasjon til å oppnå situasjonsforståelse og unngå kollisjoner. Visuelle kameraer montert på fartøyet kan brukes til å gi en detaljert beskrivelse av fartøyets omgivelser.

I denne masteroppgaven er en algoritme for objektdeteksjon i bilder basert på ikke-ledet læring implementert og testet. Ulikt ledet læring har ikke-ledet læring ingen fasit å bruke som veiledning for å lære hva slags objekter som skal detekteres. Dette sparer svært mye manuelt arbeid, og muliggjør deteksjon av objekter algoritmen ikke har sett før.

Algoritmen er basert på et nevralt nettverk, nærmere bestemt en auto-enkoder, som er trent til å representere det visuelle ved det maritime miljøet godt, samtidig som den fremhever det den anser som uregelmessig. Algoritmen beror dermed på antagelsen om at objekter er representert som uregelmessigheter i bildene.

Evalueringen viser at algoritmen på sitt beste er i stand til å detektere alle objekter. Dette kommer imidlertid på bekostning av et høyt antall falske positive deteksjoner. Det maritime miljøet inneholder mange naturlige detaljer, for eksempel bølger, som skiller seg ut visuelt. Disse regnes ofte som uregelmessige av algoritmen, og detekteres derfor som objekter.

Contents

Acronyms

AI Artificial Intelligence

ASV Autonomous Surface Vehicle

ANN Artificial Neural Network

CNN Convolutional Neural Network

GPU Graphical Processing Unit

GT Ground Truth

MAE Mean Absolute Error

MSE Mean Squared Error

PCA Principal Component Analysis

RGB Red, Green, Blue color model

SGD Stochastic Gradient Descent

Glossary

Anomaly Data inconsistent with the rest of the dataset.

Dilation The parameter deciding the jump in extracted input patch of convolutional layer.

Epoch Number of training iterations over the dataset.

Kernel size The size of the filter kernel.

Minibatch A set of instances fed at a time to a neural network during training.

Ownship The ship having the principal eye-point.

Padding The amount of zeros appended to the spatial axes of an input feature map.

Stride The parameter deciding the downsampling in a convolutional layer or upsampling in a deconvolutional layer.

Tensor A multidimensional array.

Texture Structured content that is repeatedly visible in the image.

Chapter 1

Introduction

1.1 Background and motivation

Ever since the first industrial revolution, technological progress has increased in speed and made each generation of human kind capable of achievements beyond the imagination of their ancestors. At the time, Industry 4.0 is claiming its territory, bringing increased connectivity, more available data, robots and autonomy on many levels. Autonomous vehicles play a major role within this revolution.

1.1.1 Autonomous operation

Vehicles able to operate without any human interaction have several benefits. Accidents caused by human errors are greatly reduced, leading to a large improvement in safety. Energy efficiency is optimized, reducing environmental impact. Vehicles are designed only for their purposes, not constrained by the requirement of carrying human operators. Human capital is made available for other purposes when not needed to operate the vehicle. All of these implications are highly welcome from an economical perspective. Also, autonomous vehicles can reach out in harsh environments without endangering humans, enabling search and rescue and discovery missions otherwise not possible to carry out.

Autonomous surface vehicles (ASVs) are likely to have a huge impact in the maritime industry. As over 90% of the world's trade is carried by sea [42], the potential implications are enormous. The automotive companies have been pioneers in the development of self-driving cars, and a lot of the technology is also applicable in maritime use-cases. Still, different environments must be taken carefully into account. The obvious fact that roads are still while ocean is not might make things seem easier for car manufacturers. However, the surroundings of cars at roads are quite tight and may include clutter such as pedestrians, cyclists and other cars. Vessels are likely to be surrounded by other vessels nearby harbour, but is often left with a lot of space once at open sea.

1.1.2 Sensor fusion and cameras

Sensor fusion is a key component to provide situation awareness for ASVs. Fusing information from several sensors is needed to exploit the strengths of each of them, and bring a certain level of redundancy. The visual camera is a very important sensor in this system, and has a quite different approach for detection than what radar and lidar have.

Seagoing vehicles have long used radar to detect obstacles such as vessels and shore, leading to a range of available radars specifically designed for maritime applications [19]. Ship radar provide a 360 degree field of view, accurate long-range distance measurement, robustness towards weather and illumination conditions, but lack short-range accuracy and details about detected obstacles. Newer types of radar designed for the automotive industry have better short-range performance and can hence complement regular ship radar. Lidar offer a highly detailed 3D map of the close surroundings which include accurate distance measurement and make obstacle classification possible. Its high price is an apparent drawback, and it is less robust towards weather conditions than radar since it operates in the visual spectrum.

Visual and infrared spectrum cameras also provide detailed descriptions of nearby surroundings of the vehicle. Infrared cameras are robust towards darkness and difficult illumination, which is one of the largest disadvantages of visual cameras. Yet, visual cameras are used widespread by consumers and in industry, which makes them cheap,

the technological development very fast and computer vision algorithms well-known and available. Visual cameras provide high resolution, coloured images well suited for detection and classification of objects. As opposed to other sensors do visual cameras in general capture the same information as the human eye.

About sensor selection, interesting observations can be obtained by once again looking at the development for autonomy in the automotive industry. Whereas Google has chosen to rely on lidar in addition to cameras and radar, Tesla has so far found it unnecessary to include lidar. Regardless if lidar proves to be necessary or not in a successful sensor fusion scheme for ASVs, visual cameras are likely to contribute a lot.

1.1.3 Convolutional neural networks and unsupervised learning

Deep learning based on neural networks is together with reinforcement learning at the very forefront of artificial intelligence, and is revolutionizing human society in consumer and industrial applications. Convolutional neural networks (CNNs) in particular have impacted the field of computer vision dramatically in the recent years. Ever since AlexNet [21] won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012 by reducing the localization error from 42.5% to 34.2% [37], all following state-of-the-art approaches for classification and detection are based on CNNs [2]. It is not very likely that non-learning based algorithms suddenly should outperform CNNs. Consequently, for the object detection task in this thesis, the most promising approach is likely to be based on a CNN.

The approach presented in this thesis is based on unsupervised learning. Whereas supervised learning algorithms are told what the correct outcome is in each case it is presented with, does unsupervised learning decide by itself what to learn from each situation. Supervised, state-of-the-art algorithms for object detection balancing accuracy [24] and speed [33, 34] perform great at detecting previously seen objects, but not equally good on unseen objects [49]. Unsupervised learning algorithms does not incorporate this distinguishment, and may be trained to become better at detecting unseen than seen objects. This is an advantage for an ASV application, since it needs to detect all types of objects in its surroundings, in particular novel ones.

Another benefit of unsupervised learning is that the lack of supervision makes it require far less human labor. Even though large public datasets are available [36], specific-purpose algorithms often benefit from training on domain data, which is an expensive procedure to supervise. Consequently, no supervision makes far more data accessible at a small cost. One can argue that both approaches are needed in an adequate sensor fusion scheme; the supervised algorithm is qualified to detect the most common and well-known objects, while the unsupervised algorithm is capable of detecting both novel and known objects.

1.2 Outline

This thesis is organized according to the following outline:

- Chapter 2 and 3 provides theoretical background about image processing and deep learning.
- Chapter 4 presents related work of unsupervised object and anomaly detection, with particular focus on relevant research about autoencoders.
- Chapter 5 describes the image data used to evaluate the detection algorithm and train the autoencoder the algorithm is based on.
- Chapter 6 presents the object detection algorithm and the model of the convolutional autoencoder.
- Chapter 7 describes the experiment of training the autoencoder and the evaluation results of the detection algorithm.
- Chapter 8 concludes this thesis and brings suggestions for further work.

Chapter 2

Image processing

In a computer vision perspective, image processing involves applying a filter to an image to change its properties. Image filtering is a simple process, yet very useful for several operations. The range is from fundamental tasks such as noise smoothing and edge detection to more advanced computer vision algorithms including convolutional neural networks presented in Chapter 3. The content of this chapter is based on Forsyth and Ponce [9] and Szeliski [38], and the exposition is similar to the one in [41].

2.1 Image filtering

2.1.1 Convolutional operation

In a simple form, image filtering can be described as computing weighted sums of pixel values adjacent to a centre pixel to extract information about the area. Pattern of weights is commonly referred to as the *kernel* of the filter. Filters are usually implemented by use of the convolutional operation, denoted $*$. For a two-dimensional image I of size (i, j) and a filter kernel K of size (m, n), the output image S of the convolutional operation is given by (2.1).

$$S(i, j) = (K * I)(i, j) = \sum_m \sum_n I(i - m, j - n)K(m, n) \tag{2.1}$$

2.1.2 Gaussian smoothing

Gaussian filter is a very popular approach for smoothing of images, which is useful for high-frequency noise reduction and applied in a number computer vision algorithms as pre-processing. In the two-dimensional continuous case, the gaussian filter kernel is the function describing a gaussian distribution. With standard deviation σ, it is given by

$$G(x,y) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) \tag{2.2}$$

To work with digital image data, the gaussian filter kernel is approximated to the discrete domain. An example of kernel size 5×5 with $\sigma = 1$ is shown in (2.3).

$$G(m,n) = \frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}, \quad \sigma = 1 \tag{2.3}$$

One should note that due to different approximation techniques and requirements on accuracy, there exist more than one unique gaussian filter kernel for a given kernel size and standard deviation. The effect of gaussian smoothing is shown in Figure 2.1.

Figure 2.1: Gaussian smoothing applied to an image. Top left: original image. Top right: $\sigma = 1$. Bottom left: $\sigma = 5$. Bottom right: $\sigma = 20$.

2.1.3 Edge detection

Edge detectors refer to any type of filter specifically designed to detect edges. This is particularly useful since edges are simple types of shapes that often form the basis for more complex patterns. Edges in images are built by large intensity variations within a quite small spatial domain. By computing the gradient in each of the two spatial dimensions separately, one obtain large values at horizontal and vertical edges. A Sobel filter is a simple type of filter that compute the gradient. To detect horizontal and vertical edges, the filter kernels K_y and K_x are convolved with the image.

$$K_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}, \quad K_x = K_y^T \tag{2.4}$$

A lot of images contain high frequency noise not necessarily visible to the human eye on regular print or computer screens. Edge detectors like the Sobel filter are very sensitive to this noise, yielding just noise as output if convolved directly with an image. Therefore, a better approach is to apply a smoothing filter like the gaussian before

applying Sobel. Figure 2.2 shows the response of the filter kernels K_y and K_x after the input image is convolved with a gaussian filter with standard deviation $\sigma = 10$.

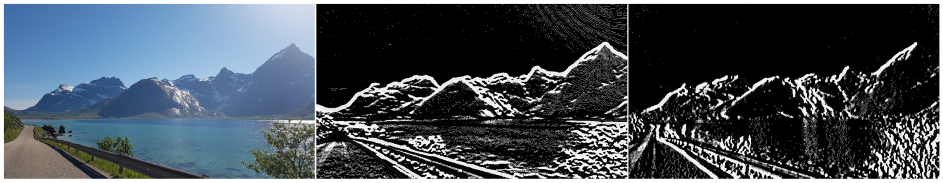


Figure 2.2: Sobel filter applied to an image. Left: original image I. Middle: $K_y * G * I$. Right: $K_x * G * I$.

Chapter 3

Deep learning

Image processing presented in Chapter 2 is fundamental to solve several computer vision tasks, both by traditional feature-based and modern learning-based techniques. Feature-based methods use domain knowledge to derive analytic models used to extract information from images. Modern, learning based techniques make general purpose models suitable to specific tasks by presenting domain data to them. Convolutional neural networks (CNNs) have achieved state-of-the-art results in tasks such as object detection and classification in images [37], and are fundamental in the object detection approach presented in this thesis. Before CNNs are described, a general understanding of deep learning is needed.

This chapter situates deep learning within machine learning and artificial intelligence, provide an overview of machine learning basics and describe artificial neural networks quite detailed. Also, autoencoders and relevant evaluation measures are presented. Apart from where cited otherwise, the structure and content of this chapter is heavily based on the book *Deep Learning* by Goodfellow et. al. [10] and F. Chollet's book *Deep learning with Python* [7].

3.1 Artificial intelligence, machine learning and deep learning

Artificial intelligence (AI), machine learning and deep learning are three closely related fields of research. A common way to distinguish them, is to define AI as the broadest term of the three, machine learning as a subfield of AI and deep learning as a subfield of machine learning. Such a separation is illustrated by the venn diagram in Figure 3.1.

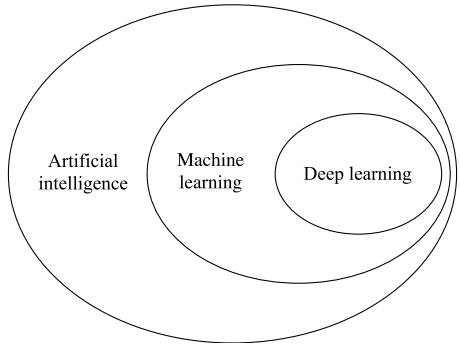


Figure 3.1: The relationship between the terms artificial intelligence, machine learning and deep learning.

3.2 Basics of machine learning

Machine learning is a subfield of artificial intelligence, recognized by its ability to learn from data rather than being programmed to follow a predefined set of rules. Machine learning algorithms are designed to find statistical structures in the data presented, in order to learn its own rules for automating the task at hand. Typical tasks for machine learning algorithms are regression, classification, anomaly detection, transcription, machine translation, synthesis, denoising and density estimation [10]. To learn rules, the algorithm needs a measure of performance of its goal, e.g. to maximize the accuracy of the algorithm.

3.2.1 Data

The data presented to the algorithm for it to learn, is referred to as *training data*. Even though statistical structures in training data might be interesting in itself, the value of well-performing machine learning algorithms is that they can be trained to do useful operations to new data, referred to as *test data*. In other words, the learning algorithm should be able to *generalize* from training scenarios to scenarios it has never seen before. This can be difficult to achieve, and a typical problem that arises is that the learning algorithm *overfits*, meaning that it fits too precisely to the training data for it generalize well to the test data. The opposite case is when the algorithm fits too loosely, it *underfits*. The best way to make a learning algorithm generalize well, is to train it on more data [7]. However, gathering data is an expensive procedure and not always feasible. *Regularization* is any modification of the learning algorithm or the data to fight overfitting and improve generalization.

3.2.2 Supervised and unsupervised learning

A broad categorization of machine learning algorithms can be obtained by dividing them into *supervised* and *unsupervised* algorithms. Unsupervised algorithms are presented with a dataset, for instance represented as a two-dimensional array of examples at one axis and corresponding attributes on the other, and learns the structure representing the data. Supervised algorithms is presented with the same dataset, in addition to an associated label for each example in the data. The supervised algorithm then learns the mapping from the attributes to the label.

3.3 Artificial neural networks

The key component of deep learning is the artificial neural network and feedforward neural network in particular, recognized by the fact that information is passed forward through the network without any feedback connections. Feedforward neural networks are quite simply general mathematical models created with the goal approximate any function. For example, a regressor described by the function $y^* = f^*(x)$ maps an in-

put x to an output y^*. By adjusting its parameters $\boldsymbol{\theta}$, a feedforward neural network $y = f(x;\boldsymbol{\theta})$ approximates the regressor. Neural networks with feedback connections are called recurrent neural networks (RNNs). Since RNNs are not used in this thesis, feedforward neural networks will simply be referred to as neural networks.

3.3.1 Building blocks

A way to describe artificial neural networks at a lower level, is to define their building blocks *neurons* being organized in *layers*. A neuron computes the weighed sum of its inputs and a bias term, typically followed by a nonlinear *activation function* which makes the network capable of modeling nonlinear phenomenons. This is visualized in Figure 3.2. More neurons with similar properties coupled together form layers, which again can be used to form the network structure shown in Figure 3.3. The weights and bias terms of the network are the parameters that are adjusted during training. *Hyperparameters* are parameters that describes the network structure at a higher level and is set before training. Examples of hyperparameters are the number of neurons in a layer, the number of layers and the activation function of choice. The process of training and more about hyperparameters is described throughout this chapter.

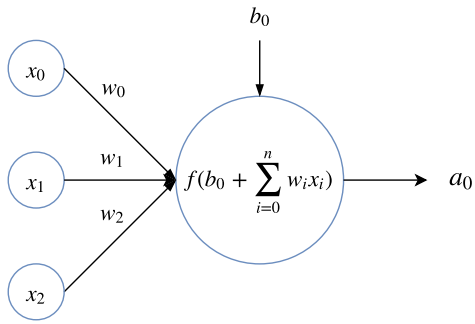


Figure 3.2: The three inputs x_i and corresponding weights w_i in addition to a bias term b_0 are passed to a single neuron. f is a nonlinear activation function, and the output of the neuron is its activation a_0.

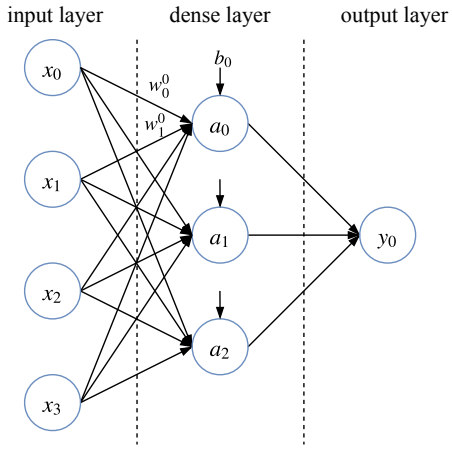


Figure 3.3: A simple neural network with a single hidden dense layer. Figure inspired by [41].

3.3.2 Layer types

A layer in a neural network is simply a way to organize neurons and how they are connected. Without any strict definition, there exist indefinitely many different types of layers. In this section, two of the most popular layer types are presented. Dense layers are a simple type, while convolutional layers form the basis for any convolutional neural network.

Dense layers

Dense layers, also called *fully connected* layers, are the simplest layer type used in neural networks. A neuron in a dense layer is connected to all activations in the previous layer. Hence, the dense layer is able to learn global patterns, which for image data means patterns built by all pixels in the input image. A major drawback with dense layers is that they are computationally expensive to train, since the number of parameters they include is very large.

Convolutional layers

Convolutional layers utilizes the convolution operation, as stated in (2.1), to learn local patterns, i.e. patterns built by the values in a small area in the input. F. Chollet [7] points out two important properties of learning local rather than global patterns by applying convolutional layers, since "the visual world is fundamentally translation invariant and spatially hierarchical".

- Patterns learned are *translation invariant*, such that a pattern learned in a specific location can be recognized anywhere afterwards. In contrast, a dense layer would need to learn the pattern again for every new location it shows at. This property can also be called *parameter sharing* [10], since the same filter kernel (with the same parameters) is used all over the input image.

- Patterns learned are ordered in *spatial hierarchies*, in the sense that the first layers learn simple patterns like edges, and the following layers learns increasingly complex patterns.

For images, learning local patterns is obtained by applying filter kernels in a similar way as in standard image processing, however the difference lies in how the parameters of the kernel are decided. In standard image processing, fixed, handcrafted filter kernels are used to obtain a certain property, for instance detection of vertical edges. In convolutional layers the kernels consist of parameters (weights) that are subject to change during training, such that the model learns its own filter kernels. There are only practical limitations on how many filter kernels a convolutional layer can have.

A convolutional layer is described by four hyperparameters:

- The filter size K decides the spatial receptive field of each activation in the output. Larger K also increase the number of parameters involved with the layer.

- The number of filters F. Larger F increase the number of parameters and decides the output depth.

- The stride S decides how many pixels the center of the extracted patch is displaced while sliding over the entire input. $S > 1$ effectively downsamples the spa-

tial extent of the output.

- The amount of padding P decides in any extra rows and columns with zeros should be added around the input. A common choice is to zero pad such that the spatial dimension of the output of the layer is the same as its input.

Figure 3.4 visualizes a convolutional layer in detail. The input feature map has dimension $5 \times 5 \times 3$, corresponding to a very small RGB image. The input is padded with zeros such that $P = 1$, and the centre of patches extracted are displaced such that the stride $S = 2$. The filter kernel of size $K = 5$ and number of filters $F = 4$ is applied to the input by extracting patches from the input and computing the dot product. The given hyperparameters yield an output feature map of dimension $2 \times 2 \times 4$.

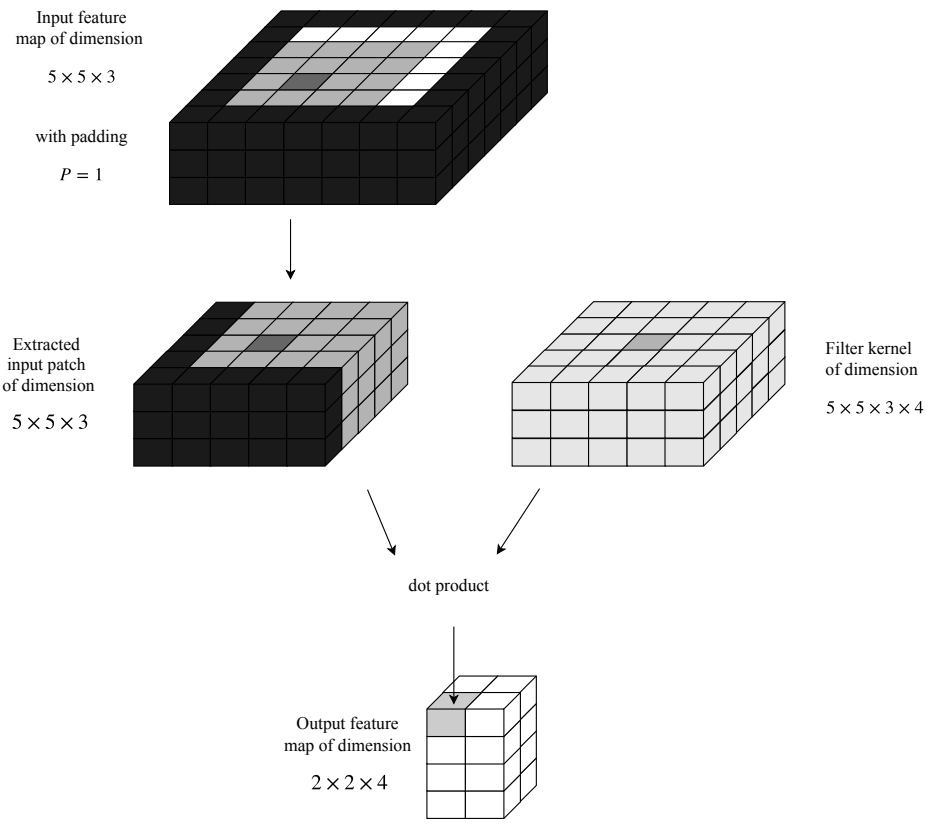


Figure 3.4: Convolutional layer. The input patches are extracted with a stride $S = 2$. The number of filters $F = 4$ yields an output feature map of depth 4. Figure inspired by [7] and [41].

3.3.3 Training a neural network

The process of adjusting the parameters of a neural network to obtain a certain goal, is called *training*. The goal is formulated as a optimization problem, such that the network seeks to minimize the loss function. Consider a simple regression problem with multi-input $\boldsymbol{x}$ and single-output y. We choose the goal to be to minimize the mean square error between the predicted output $\hat{y}$ and the ground truth output y over all the n instances in the training data. For a prediction $\hat{y}_k = f(\boldsymbol{x}_k; \boldsymbol{\theta}), k \in [1, n]$, the loss function is given by

$$L = \frac{1}{n}\sum_{i=1}^{n}(f(\boldsymbol{x}_i;\boldsymbol{\theta}) - y_i)^2 \tag{3.1}$$

A linear regression problem with relatively few data instances could easily be solved for where the gradient of the loss function is zero, $\nabla L = 0$. However, due to the non-linear activation function often used in neural networks, most interesting loss functions become non-convex. The common solution to this is to use an gradient descent optimizer to iteratively train the network to lower the loss function. While a very small dataset allows for training by regular gradient descent, this soon become infeasible for larger datasets due to the limited memory of computers. *Stochastic gradient descent* optimizers solves this by calculating the gradient descent of a *minibatch* of instances at a time, including anything from one to a larger number of instances. Hence, the gradient is estimated from a relatively small number of instances. Including all the instances in the training set in a minibach would correspond to regular gradient descent.

The examples in each minibatch are randomly selected. Training on the entire training set once is called an *epoch*, and since the training is iterative, the network may require training for several epochs to obtain satisfying performance. For each epoch, new minibatches of randomly selected selected examples are created to avoid the same learning pattern as in the previous epoch. The randomness is important for generalization since it introduce noise.

The gradient descent scheme is given in (3.2), where $\boldsymbol{g}$ is the gradient and α is the *learning rate*.

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha\boldsymbol{g} \tag{3.2}$$

With the loss function in (3.1), the gradient is approximated from the minibatch of size n'.

$$\boldsymbol{g} = \frac{1}{n'}\nabla_{\boldsymbol{\theta}}\sum_{i=1}^{n'}(f(\boldsymbol{x}_i;\boldsymbol{\theta}) - y_i)^2 \tag{3.3}$$

Neural networks may have many hidden layers. The gradient used for parameter update of the last layer is computed from the loss with the scheme given above. However, this is not the case for the hidden layers since they output an activation that cannot

compared directly with a ground truth value.

Back-propagation [35], introduced by Rumelhart et. al. in 1986, is an algorithm for computing the gradient of any function. Starting out from the loss of the last layer, the gradients of the hidden layers are computed by recursively applying the chain rule for gradients. Even though other alternatives exist, back-propagation is the standard approach for gradient computations in deep learning due to its computational efficiency.

Optimizers

An optimizer is the overall scheme of the optimization process when training a neural network. The most successful ones are varieties of stochastic gradient descent, described by (3.2) and (3.3). The learning rate α in (3.2) is a very important hyperparameter which can be difficult to set manually. Following, several optimizers computes adaptive learning rates for different parameters individually [47, 20]. Another common approach used by optimizers is to add *momentum* [31], in which a moving average of the last gradients is used and not only the current one.

3.3.4 Regularization

Regularization refers to any modification of the original neural network and its training to increase generalization and reduce overfitting. This is a vast field of research within deep learning. The best way to increase generalization is to train the model on more data, but gathering data can be an expensive exercise. Therefore, common modifications involve to reduce the network capacity, add terms in the loss function, augment the input data by various transformations and to make sure the network does not train too long. Table 3.1 shows a selection of regularization techniques with a short description. Dropout and batch normalization is explained in more detail in the following sections.

Table 3.1: Selection of common regularization techniques.

	Description
Early stopping	More training make the model fit better to the training data. To fit well to the test data, training should by stopped at the right time.
Reduce model capacity	Reduce the number of trainable parameters by reduce the number of neurons and layers and change layer type. Fewer parameters can describe less complex patters, thereby less likely to overfit.
Weight decay	Add terms in the loss function penalize large weight parameters. Weights closer to zero improves generalization.
Data augmentation	For image data, applying transformations such as rotation, translation, shear and zoom with random, but controlled strength generates realistic new data from existing data.
Dropout	Randomly set activations to zero. The technique is elaborated below.
Batch normalization	Adaptively normalizing each minibatch of values keeps the input distribution of a layer fixed, which both speeds up training and acts as regularizer. The technique is elaborated below.

Dropout

A very powerful regularization technique called *dropout* introduced by Srivastava et. al. [40] has become common for training neural networks due its great performance and simplicity. When dropout is applied to a layer, the activations of neurons are randomly set to zero, such that they are "dropped out" from the network. The dropout rate p defines the fraction of the neurons in a layer that is set to zero. Figure 3.5 shows an example of dropout applied to an activation matrix.

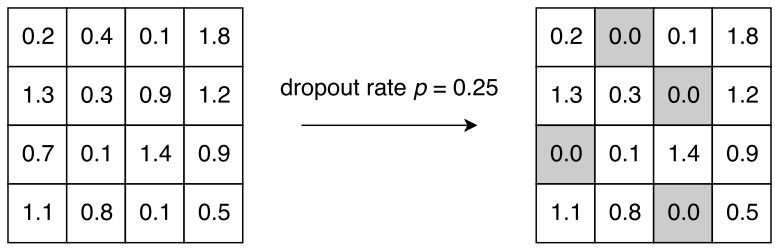


Figure 3.5: Dropout applied to a feature map with the dropout rate $p = 0.25$. Figure inspired by [7].

Applying dropout to a neural network leads to the following:

1. During training, the activations are randomly set to zero in the layers where dropout is applied. This corresponds to sampling different "thinned" networks for each training case.

2. During testing, no activations are set to zero. Instead, and to make up for the dropped-out activations during training, all activations are scaled down according to the dropout rate, such that the expected output the same as during training.

As pointed out in [40], how dropout affects the network can be interpreted in different ways. Firstly, it can be seen as randomly sampling sub-networks of the original network and averaging the output of all the sub-networks during testing. This is similar to combining an ensemble of models, even though one should note that the models in the dropout-case share parameters and are thus not independent. Combining models has shown to nearly always improve performance of machine learning models [40]. Secondly, dropout can be seen as a way of introducing noise to the particular layer it is applied in. Random noise can break up patterns present in the data but not in the process the data samples from, e.g. due to sampling bias and non-random noise. Effectively, dropout acts as a regularizer such that the neural network is less likely to overfit and more likely to generalize well to new data. It is computationally efficient, and works well on nearly any model that is trained by stochastic gradient descent-based optimizer [10].

3.3.5 Optimization strategies for training

Whereas optimizers are algorthims that applies to the overall training process, other strategies that optimize training on a lower level exist. Normalization techniques are one of those, with batch normalization as the most influential so far.

Batch normalization

A well known fact of neural network training is that normalizing the input speeds up the learning process [23], since it keeps the distribution of the input fixed during training.

Assuming the data is Gaussian distributed, the normalization is done by subtracting the mean to center the data around zero, and to divide by the standard deviation to obtain unit variance. For a one-layer neural network, normalizing the input of the first and only layer is sufficient. However, for a deep model with several layers, it is advantageous if one could ensure that the input of all layers have fixed distribution.

During training of a deep neural network, a minibatch of training examples is usually considered at each step, and the trainable parameters of each layer are updated according to the gradient descent of that minibatch. By use of back-propagation, the gradient computation propagates from the last layer to the first layer. A small change in the parameters in the last layers may in some cases be amplified throughout the deepness of the network, creating a relatively large change in the activation in the first layers. This may lead to a significant change of input distribution of the last layers.

Ioffe and Szegedy [17] introduced *batch normalization* in 2015 to address this problem. Consider a batch of images defined by a four dimensional tensor with dimensions (N, H, W, C), where N is the number of images in the batch, H and W are the spatial axes and C is the channel axis. For each channel, normalization along the (N, H, W) axes are done by computing the mean and variance of those axes together, and reparametrizes the values such that the distribution in each channel stays fixed [44, 10]. Now, each layer can assume a certain channel distribution, which allows for higher learning rate and deeper networks than what was feasible from before. Batch normalization has also proven to act very well as regularizer since the reparametrization introduces both additive and multiplicative noise. This effect is so evident that dropout either can be reduced in strength or removed entirely.

Recent research shows that there are drawbacks of normalizing along the batch axis in certain scenarios. Very small batch sizes lead to less accurate mean and variance estimation than for larger batch sizes, which reduces the original effect of batch normalization. In those cases, alternative normalization methods should be considered. *Group normalization* by Wu and He [44] normalizes for each sample N along all the spatial axes and a subset (a group) of the channels, computing the mean and variance within that subset. Wu and He show that group normalization has performance that

almost can compete with batch normalization in most regular scenarios, while it outperform batch normalization for very small batch sizes.

3.4 Autoencoder

An autoencoder is a neural network with the overall goal to copy its input to its output. It consists of two parts: an encoder and a decoder. The encoder e encode the input $\boldsymbol{x}$ to a hidden representation or code $\boldsymbol{h}$, while the decoder d decode the hidden representation to the output or *reconstruction* $\boldsymbol{r}$. Thus, $\boldsymbol{r}$ is given by (3.4) and the overall structure of the autoencoder is visualized in Figure 3.6.

$$\boldsymbol{r} = d(\boldsymbol{h}) = d(e(\boldsymbol{x})) \tag{3.4}$$

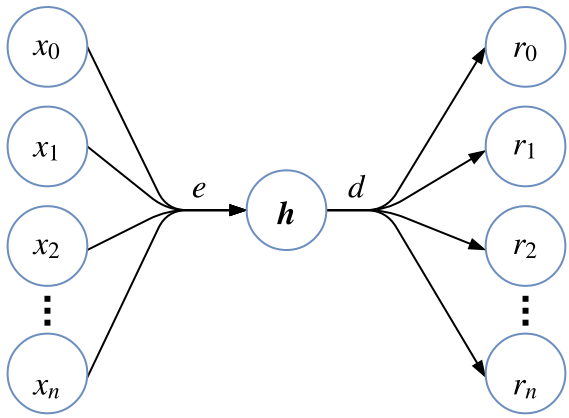


Figure 3.6: Autoencoder with input dimension n. The encoder e encodes the input to the hidden representation or code $\boldsymbol{h}$ of arbitrary dimension and the decoder d decodes the hidden representation to the reconstruction r.

Autoencoders are trained in the same way as any other feedforward neural network, but uses its input as ground truth and seeks to minimize the difference between input and output. Since the training procedure do not require any manual labeling of ground truth, autoencoders may be regarded an unsupervised learning algorithm [10], a view adopted in this thesis. Another option [7] is to consider autoencoders as self-supervised, in the sense that they supervise their own learning with their input.

By constraining the network and hidden code in various ways, the autoencoder can learn useful aspects about the input data. For instance, if the hidden code is of lower dimensionality than the input, the trained autoencoder acts as a compression algorithm since it is forced to learn the most important properties of the input. Such an autoencoder architecture may yield a hidden code containing similar information to the output of principal component analysis (PCA); they are both a low-dimensional representation of the input. However, there are several differences: PCA finds linear combinations of the original dimensions and outputs the dimensions that explain the greatest variance, by including the exact variance each dimension explain. Autoencoders does not yield a similar ordering of its dimensions based on the information it contain. By the use of nonlinear activation functions, autoencoders capture patterns that PCA is incapable of. This make them serve as a "nonlinear generalization of PCA", and yield far better representations in many applications [13]. In addition to dimensionality reduction, numerous other applications of autoencoders exists, and a selection of relevant examples is reviewed in Chapter 4.

3.4.1 Denoising autoencoder

A denoising autoencoder is a variety of an autoencoder in which the input data is corrupted with noise, yet it is trained to reconstruct the non-corrupted version of the input [10]. Thus, the autoencoder should become more robust towards the particular type of noise added than what originally is the case. Additionally, it can be applied to the computer vision task known as image restoration, where the goal is to restore the input image with increased quality.

Common noise choices are random noise such as Gaussian or uniform distributed noise, which share characteristics with dropout (Section 3.3.4) applied to the input layer. Other examples are text and larger patches of $n \times m$ pixels that is dropped out. For the latter, the task of denoising transforms to the closely related task of inpainting or synthesis: the autoencoder learns to fill in the masked area based on context of nearby pixels.

3.5 Evaluation measures

3.5.1 Binary classification

Sample space

In binary classification problems, the four outcomes of true positive, false positive, true negative and false negative define the sample space [32]. They can be expressed in terms of raw counts or in relative terms such as probabilities or proportions. Table 3.2 summarizes how the outcomes are distinguished.

Table 3.2: Contingency table summarizing the sample space in binary classification problems. Courtesy of [22].

	True condition postive	True condition negative
Predicted condition positive	True positive *tp*	False positive *fp*
Predicted condition negative	False negative *fn*	True negative *tn*

Precision and recall

Precision[1] and recall[2] are evaluation measures based on the outcomes given in Table 3.2. Precision may be interpreted as the fraction of how many selected items are relevant, while recall is how many relevant items are selected. Their definition is given in (3.5) and (3.6)

$$\text{precision} = \frac{tp}{tp + fp} \tag{3.5}$$

$$\text{recall} = \frac{tp}{tp + fn} \tag{3.6}$$

[1] Precision is also known as confidence or True Positive Accuracy.
[2] Recall is also known as sensitivity or True Positive Rate.

Chapter 4

Related work of object and anomaly detection

Unsupervised object detection in computer vision applications by the use of neural networks is a quite recent field of research, with the first approach published in 2017 based on an autoencoder [15]. However, the related field of unsupervised anomaly detection has been studied extensively [6, 14, 30], with autoencoders as one of several methods. If one assumes that the objects one wish to detect are represented as anomalies in the data, anomaly detection approaches can be used to detect objects in images. This is similar to the methodology followed by [15].

This chapter provides an overview of unsupervised detection approaches based on convolutional neural networks.

4.1 General detection approach with autoencoder

Detection of anomalous features in images by the use of autoencoders is a well studied approach based on unsupervised learning [22]. The general procedure is given by the two following steps:

1. Train the autoencoder as a feed forward neural network on data imaging normal

scenarios minimizing the pixel wise reconstruction residual.

2. Apply trained autoencoder on new images and compute the corresponding reconstruction residual. Classify as normal or anomalous depending on the residual magnitude.

Several varieties of both steps exist. The main discussion of the first step concerns about what data the model should see during training, and is a discussion for anomaly detection in general. Consider a dataset consisting of two classes: a normal class and an abnormal class, but without knowledge of what class each data point belong to. A common assumption is that the normal class is well sampled, while the abnormal class is severely under-sampled, due to the nature of abnormalities [30]. This data imbalance often leads to a under-sampled class with high variance and skew distribution, which must be taken into account when designing any learning algorithm, both with and without supervision [12]. The problem can be mitigated by taking steps to ensure that the abnormal class is even less well sampled than originally, and then simplify by treating the entire dataset as all belonging to the normal class [6]. Such a simplification is in particular suitable for unsupervised approaches.

How to select a threshold for what is normal and what is not is of focus in the second step. Fixed thresholds is the most general approach, and can easily be tuned for the algorithm to yield the desired ratio of true detections, false detections and missing detection (precision and recall). Another option is to base the threshold on a moving average of the pixel values of the set of images, as adopted by [15]. This requires to algorithm designer to make an assumption on the sparsity of detections in the feature map, in addition to set values for two more parameters.

4.2 Detection with inpainting autoencoder

Following Section 4.1, most former autoencoder for detection train to simply minimize the residual between input and reconstructed image. Very recent work by Bhattad et. al. [5] argue that this is a naive and too simple approach. The idea behind using an autoencoder for detection is that when it is applied to a new image at test time, the au-

toencoder should reconstruct the typical image from its training that is the most similar to the new image. What often happens is that the autoencoder "peeks" and copies its input image directly through, although with altered resolution. This leads to a small reconstruction residual even for obvious anomalous features, and thus a low detection rate.

The proposed solution by [5] is to use a variety of a denoising autoencoder, namely an inpainting autoencoder. The noise added to the input image of an inpainting autoencoder is relatively large patches masking an area of pixels in the image. During training, the autoencoder is forced to learn how to inpaint the area being masked considering the context of adjacent pixels. The final output consists of a grid on inpainted areas merged together to a complete image. This approach ensures that "peeking" is no longer possible. The autoencoder is still trained to minimize the reconstruction residual, and the reconstruction residual is used as feature map for anomaly detection, following the same idea as regular detection autoencoders.

To test their anomaly detection approach, [5] perform experiments on images of celebrity faces, in which the anomalous images are given by photoshops, extreme makeup and extreme facial expressions. First, tightly cropped face images are obtained using the Viola-Jones face detector [43]. Then, each image is classified as normal or anomalous based the L-infinity norm of the reconstruction residual, i.e. the localization of the anomalies are disregarded. Their results show that the no-peeking autoencoder obtain better recall rate than the regular autoencoder based on reconstruction residual, ranging from twice the performance to about equal performance. For a different application, it is straightforward to localize anomalies by applying thresholds instead of the L-infinity norm, as given in step 2 in Section 4.1.

The domain difference between celebrity faces and objects in maritime environments is significant. Bhattad et. al. points out that anomaly detection of faces is a suitable test domain due to the high level of details and that anomalous faces look quite similar to typical faces. In comparison, a flat ocean surface is far less dense of details, and vessels look quite different than ocean. However, the conditions in maritime environments are highly varying. Harsh weather conditions may introduce a high level of

details such as waves, foam, water in the air etc. that needs to be well reconstructed to not be detected as false alarms. Additionally, vessels and other objects should be detected not just within a short range. At longer distances, it might be very difficult to distinguish objects from ocean features - objects may cover only a few pixels. Consequently, the different domains might share enough properties to make a detection approach be suitable for both, even though this can only be determined by conducting proper experiments.

4.2.1 State-of-the-art inpainting

Inpainting, as a subtask of image restoration, refers to fill in an area being masked in the input and is an ancient art in itself [4]. The difficulty of the problem is very dependent on the size of the masked area, since the pixels near the boundary have less ambiguity than the ones in the middle of the mask. Recent research based on convolutional encoder-decoder (autoencoder) architectures have lead to great progress.

Training loss

Bhattad et. al. [5], discussed in detail in Section 4.2, points out that they explore an inpainting problem similar to the one in Pathak et. al. [29]. Pathak et. al. proposed an contextual autoencoder that is trained to minimize two types of losses: the pixel wise reconstruction loss and an adversarial loss, the adversarial loss as given by generative adversarial networks (GANs) [11]. [29] points out that the pixel wise reconstruction loss alone make the reconstructed image look blurry, while adding the adversarial loss leads to sharper, more realistic looking reconstructions. This is confirmed by Iizuka et. al. [16] and Yu et. al. [46], who follow up the work of [29] by proposing a more general autoencoder architecture and a GAN, respectively, for increased quality and realistic-looking inpainting. They both show improved performance over [29]. Additionally, [46] suggest improvements to the model of [16] by weighting the pixels near the boundary more in the pixel wise reconstruction loss, which speeds up training significantly.

However, to detect anomalies in images it might be beneficial to obtain a blurry reconstructions, hence why [5] choose to use pixel wise reconstruction loss only. A blurry

reconstruction make the poorly reconstructed objects stand out in the residual. A reconstruction with generated details that is only a few pixels shifted spatially to the input image might cause large reconstruction residuals at both normal and anomalous features due to the generated details. Also, it makes sense to not use a weighted loss, since this would facilitate better reconstructions and and thus lower detection rate in some areas than in others.

Mean absolute error (MAE) and mean squared error (MSE) are common choices of training loss to minimize the pixel wise reconstruction error. Of the discussed inpainting autoencoders in this section, MAE is used by [5] and MSE is used by [29, 16]. Both types were tested by [29] without experiencing any significant difference.

Architecture

The architectures of the mentioned autoencoders share the general characteristics of convolutional neural networks, but have several differences. An important attribute of the architecture of inpainting autoencoders is the size of the *receptive field*. The receptive field is the area of the input image used to compute an output pixel, and it needs to be larger than the mask for the mask to be inpainted. If the receptive field is slightly larger than the mask, only the very local context will be used for inpainting. If the receptive field is far larger than the mask, more global context is available to be used for inpainting. This is illustrated in Figure 4.1.

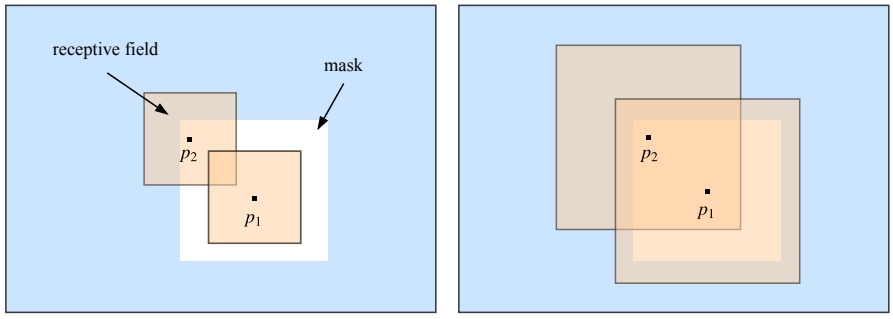


Figure 4.1: Left: the receptive field is smaller than the mask, hence p_1 cannot be inpainted. Right: the receptive field is slightly larger than the mask, such that local context can be used for inpainting. Figure inspired by [16].

While a convolutional layer have a receptive field equal to its kernel size, a fully connected layer has global receptive field since each neuron is computed by all pixels in the input image. [5] use a fully connected layer in the bottleneck of their autoencoder and [29] use a channel-wise fully connected layer to obtain the same global receptive field but with far less parameters. [16] use only convolutional layers and thereby obtain a network architecture applicable for arbitrary image sizes. They obtain a large receptive field by using several dilated convolutional layers [45], which uses kernels that are spread out over a larger area without increasing the number of parameters. As an example, dilated convolutional layer with kernel size 5 and dilation rate 2 has the same receptive field as a regular convolutional layer with kernel size 9, since every second pixel of the extracted input patch is skipped.

Another difference worth pointing out is that [5] use bilinear interpolation to upsample, while [29, 16] use transposed convolutions. Transposed convolution, sometimes referred to as deconvolution [48], is a convolution that applies a backwards stride, allowing the layer to upsample spatially. The properties of regular convolution including learnable filters and nonlinear representations are thus the same for transposed convolutions. Transposed convolutions are commonly used for dense predictions such as image segmentation [39, 18] and image restoration [26, 8].

Chapter 5

Image data

The image data used to develop an object detection algorithm must be acquired and selected very carefully. The data used to measure the performance of the algorithm, the test data, should contain a representative sample of the scenarios the algorithm will be applied in. As an example, and object detection algorithm designed for a seagoing ASV *could* be tested on images acquired by a car driving on the road, but this would not give a reliable measure of performance at sea, and is therefore largely meaningless.

Learning based algorithms such as neural networks behave according to how the algorithm is designed, in addition to how and on what data it is trained. To achieve the best possible performance on the test data, the algorithms are typically trained on similar images, for instance taken by the same cameras in the same environments, as done in this thesis. By further selecting training data wisely, one can influence the algorithm to perform as desired in chosen scenarios.

This chapter describes how the data used in this thesis is acquired in addition to provide qualitative and quantitative details about the training data.

5.1 Data acquisition

Kongsberg Seatex has formed a collaboration with Hurtigruten to collect data from vessel voyages. *MS Polarlys*, one of Hurtigruten's vessels, was selected for this purpose. In

order to get a full field of view, a camera rig of 12 cameras was mounted on four locations on the vessel, each with an angle of view of 60°. The camera rig on the vessel seen from above is illustrated in Figure 5.1. An example of the image data they acquire at a given point in time is shown in Figure 5.2.

One should note that all images, except for the ones taken by camera 5 and 11, show parts of the ownship in the very foreground, at times including passengers. Any part of this foreground should obviously not be detected as objects that are fed to the collision avoidance system of an ASV. Details describing how image data was acquired are summarized in Table 5.1.

Table 5.1: Details of data acquisition

Dates of origin	October 2017 to April 2018
Continuous route	Bergen-Kirkenes-Bergen, 33 port docks one way
Route duration	12 days
Camera model	Axis Q3708-PVE
Number of cameras	12
Frame rate	Every second

As given in Table 5.1, all cameras capture an image every second. However, the available dataset does not include images captured at every second for half a year consecutively. On average, eight hours of imaging is saved from each day. Additionally, due to a malfunction in the camera rig on the starboard side, no images from camera 4, 5 and 6 was saved from early March until end of April.

5.2 Data details

The image data is organized according to the point in time the image is taken, hereby defined as its *timestamp*. Several attributes associated with each timestamp is saved in addition to the 12 images themselves. The most important attributes include the position and velocity of the vessel in addition to a list of other vessels nearby targeted by their AIS signal[1]. An overview of image and timestamp details is given in Table 5.2.

[1]Automatic identification system. The International Maritime Organization (IMO)[1] require all ships over a certain size, depending on the ship type, to use AIS transponders. In other words, all vessels, in particular

Table 5.2: Details and attributes of images and timestamps.

Resolution	1920 × 2560 pixels
Channels	RGB
Representation	8 bit, 0-255
Format	jpg
Total number of timestamps	5,507,949
Associated attributes	Position, velocity, AIS targets

5.3 Data exploration

A lot of knowledge about the data can be gained by observing the data itself in addition to the distribution of interesting variables. This is a vital step for in advance of designing any learning based algorithm. Randomly selected examples of images given in Figure 5.3 show a diversity that motivates for further exploration. To reduce the computational complexity when estimating distributions, a subset of timestamps is obtained by only including a timestamp every 30 minute, resulting in subset of 3060 timestamps.

5.3.1 Speed of vessel

The speed of the vessel is interesting because it is a very simple way to determine in what stage of the voyage the vessel is. As clearly shown in the raw data images in Figure 5.3, the dataset include images from all stages of voyages, both when the ownship is docked and when at open sea. Images captured when docked and at open sea contain very different features, which can be utilized for selecting what features the algorithm is trained on. Instead of matching vessel positional data with the location of ports, the speed of the vessel is used to categorize if the vessel is in harbour or at sea. Figure 5.4 shows the univariate distribution of the speed of the vessel as a histogram, illustrating a very clear distinction between being docked with zero speed and in cruise speed. If one wish to obtain a dataset that only contain images from when the ownship is at a distance from harbour, timestamps from when the speed is less than 6 m/s can be removed. This yields a subset that is about 21% smaller than the original set.

small ones, do not necessarily use AIS transponders.

5.3.2 Illumination intensity

The variation in illumination and thereby visible features is large in the image data. Hence, details of when images are captured and the illumination intensity is needed. The illumination intensity is computed as the mean of all pixel values in images taken by camera 11. Only camera 5 and 11 is suited for this purpose since these are the only ones not corrupted by lights from the ownship. Camera 5 is not used due to the malfunction mentioned in Section 5.1.

The distributions of at what time of day the images are captured and the mean illumination intensity is given in Figure 5.5. In Figure 5.5a, the distribution is approximately uniform, with slightly more images captured in the hours of early morning and evening than in the hours at mid day. While this is a reasonable approach for acquiring images in general, recall that the images are taken far north and mostly during winter. Consequently, the sun is set a large part of day. The distribution of illumination given in Figure 5.5b show two quite distinct groups with mean illumination intensity of 30 to 60 and 90 to 140, respectively. Intuitively, the two groups indicates that most images are captured either in darkness at night or in well-lit conditions at day.

Image selection

Considering the fact that the images taken at night are too dark to show any interesting image features, the number of dark images is quite high. A less skewed dataset is obtained by careful selection. Of the images with mean illumination below 50, half of the corresponding timestamps are selected randomly to be removed. Removing all images with certain characteristics might be detrimental, since one cannot know exactly the content of the removed images. Removing half randomly is a safer choice and yield a more balanced dataset. This filtering reduces the size of the dataset with 20%. The resulting distributions of the hour of day images are captured and the illumination intensity is given in Figure 5.6. Clearly, the distribution of illumination in Figure 5.6b include far less examples of the first group. As a result, the filtered subset contain more images captured in daylight than in darkness. Also, it is evident in Figure 5.6a that the removed examples was mainly captured at early morning and in the evening.

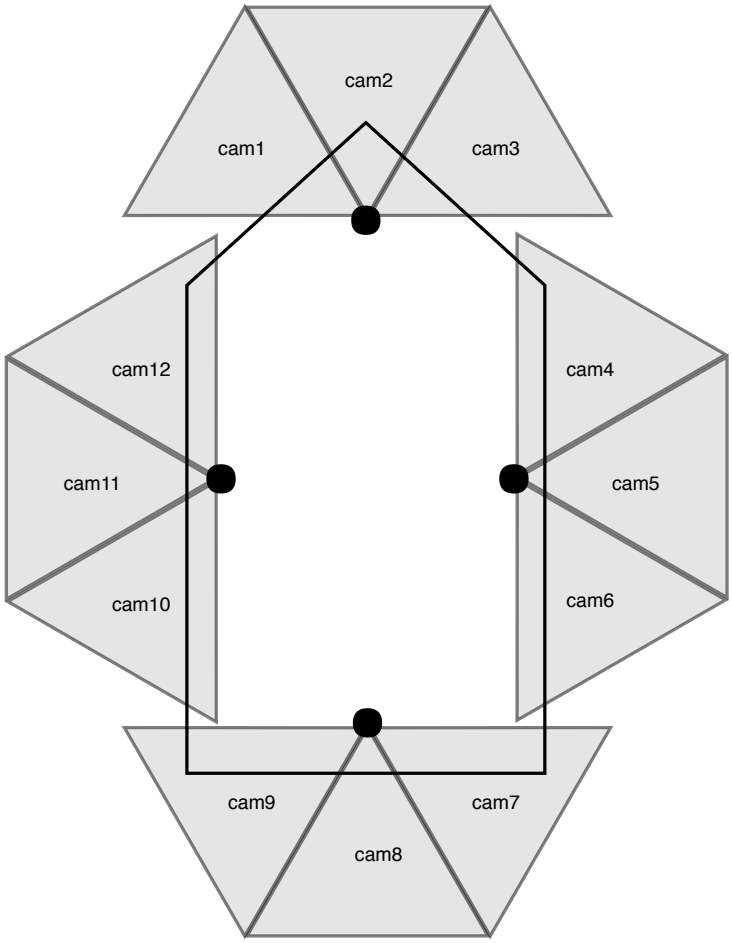


Figure 5.1: Schematic overview of camera rig on the ownship. Black dots mark where cameras are mounted. Gray areas show the approximate field of view of each camera.

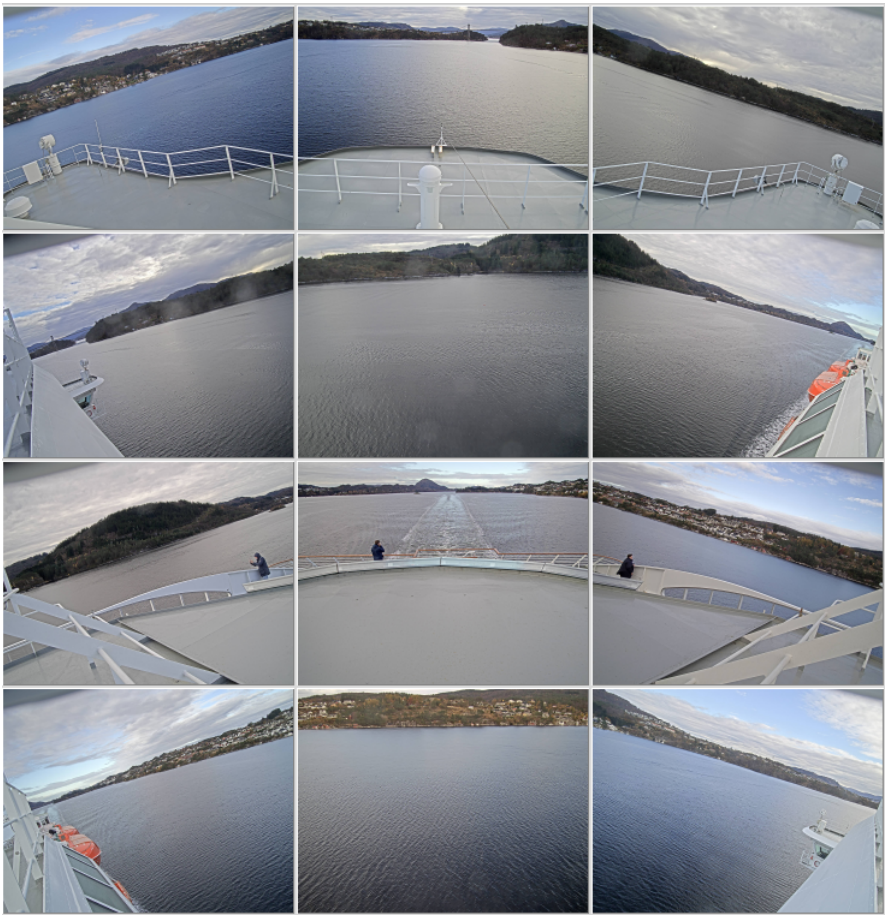


Figure 5.2: Example of all images taken at a given timestamp. The top row shows camera 1-3, the second shows camera 4-6, the third shows camera 7-9 and the bottom shows camera 10-12. Images taken by camera 5 and 11 are the only ones without parts of the ownship in the foreground.

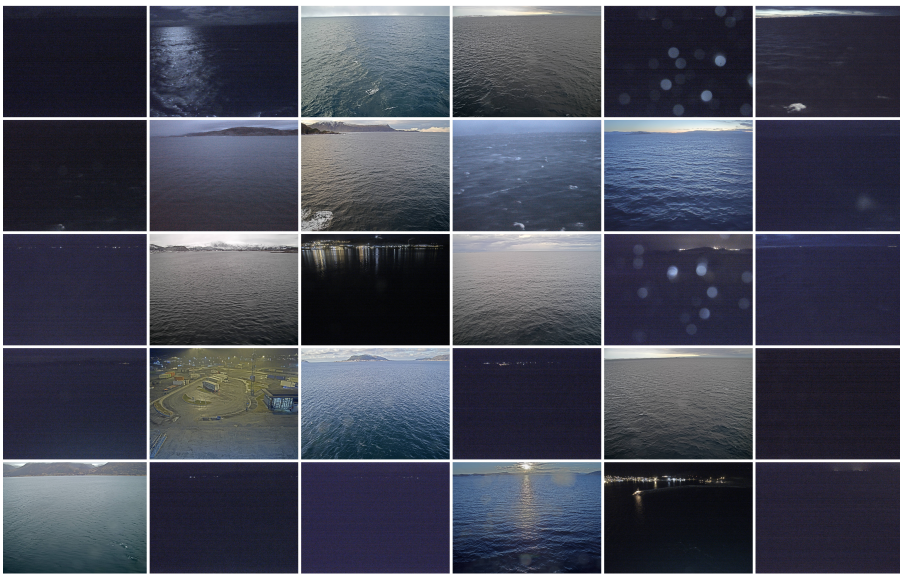


Figure 5.3: Randomly selected examples of images showing some of the diversity in the image data. Details are visible by the use of a PDF-viewer and zoom. A lot of the images are very dark, some of them are captured when the ownship is docked and some of them when at open sea. The dark images vary from complete darkness to including visible features such as lights in the horizon originating from shore or a vessel, reflections in the ocean surface from the light sources, reflections in water drops on the camera lens and waves and foam close to the ownship. The images captured in daylight include varying degree of waves, sea spray and reflections in the ocean surface, in addition to varying presence of shore.

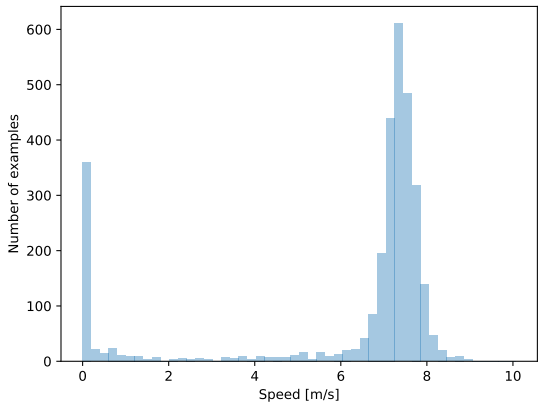
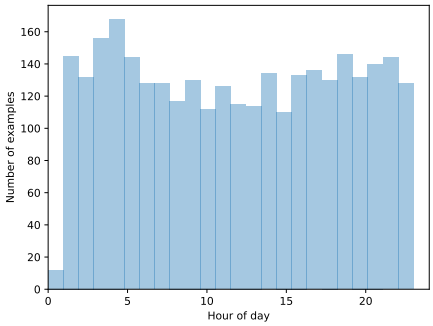
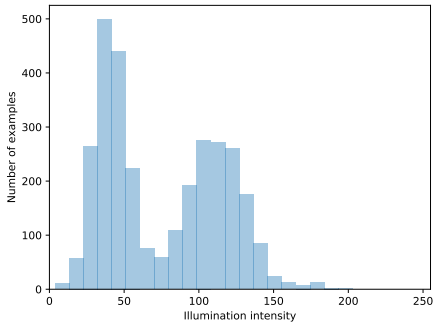


Figure 5.4: The univariate distribution of the speed of the vessel based on the subset sampled every 30 minute. Almost at all times, the vessel has zero speed or more than 6 m/s.

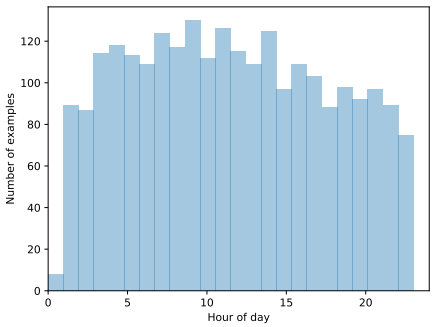


(a) The distribution is uniform, except for a low number captured the very first hour of the day. Slightly more images are captured in the early morning and in the evening than in the middle of the day.

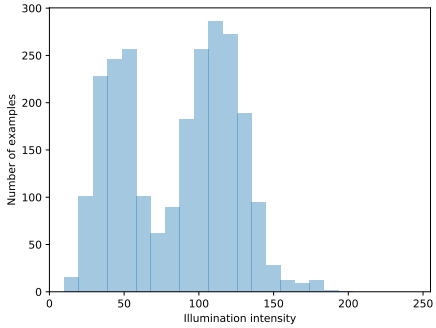


(b) The distribution shows two groups. The first with mean illumination intensity of 30 to 60 and the second with 90 to 140. Assuming they are Gaussian distributed, the first group has lower standard deviation than the second.

Figure 5.5: The univariate distribution of the hour of day images are captured and the mean illumination intensity of the original subset.



(a) The distribution is still uniform, but with opposite trend than with the original subset.



(b) The distribution still shows two groups, but now the peak of the first group is reduced.

Figure 5.6: The univariate distribution of the hour of day images are captured and the mean illumination intensity of the reduced subset with half of the examples with mean illumination intensity below 50 removed.

Chapter 6

Object detection algorithm

This chapter describes the object detection algorithm implemented and tested in this thesis. The overall algorithm is presented, in addition to details about the inpainting autoencoder that takes part in the algorithm.

6.1 General approach

The general approach of the object detection algorithm follows the work presented in Chapter 4, by using the reconstruction residual of an autoencoder as detection signal. The reconstruction residual is obtained following the two-step procedure described in Chapter 4.1. Among the related work, the anomalous face detector proposed by Bhattad et. al. [5] is of particular interest since it is designed to avoid the direct copying of input to output yielded by regular autoencoders which make objects less likely to be detected. This was also found to be a challenge in [22]. The object detection algorithm developed in this thesis follows their work with certain modifications.

6.1.1 Assumption and performance implications

A highly decisive assumption is needed if an anomaly detector can be applied to detect objects. *The objects one wish to detect are represented as anomalies in the data.* If this

assumption holds, objects such as vessels, small rocks, seamarks, beacon lights and other unknown objects can be detected since they visually stand out in the ocean.

If most of the mentioned objects are detected, the number of false negatives will be low, leading to a high recall. A high recall is arguably the most important performance metric for the detection approach in this thesis, since it tells the ratio of true objects that are detected vs. undetected. The most dangerous scenarios for an ASV arise if true objects are not detected.

On the other side, there is a clear cost of detecting objects represented as anomalies. Numerous features one do not wish to detect might also be represented as anomalies in the data, thus detected by the object detector. Examples of such features are waves, sea spray, reflections from sun and other light sources. This may lead to a high number of false positives, thereby a low precision. Obviously, a very high number of false positives is not desirable. However, it might be a reasonable price to pay if it allows the recall to be high.

6.2 Algorithm walk-through

A convolutional autoencoder is trained to inpaint masked areas of its input image by minimizing the pixel wise difference between the original image and the reconstructed image. The training data is selected to ensure that it mainly show ocean features and few objects. This facilitates the autoencoder to inpaint ocean more accurately than objects. The purpose of this is to make the residual larger for objects than for ocean, as visualized in Figure 6.1. A grid of m rows and n columns is set as a hyperparameter before training and decides the size of the mask. During training, a minibatch $\mathbb{B}$ of images is considered at each iteration. Each image in the minibatch is masked at different areas randomly selected of the $m \times n$ different masks, yielding the masked minibatch $\mathbb{B}_m$.

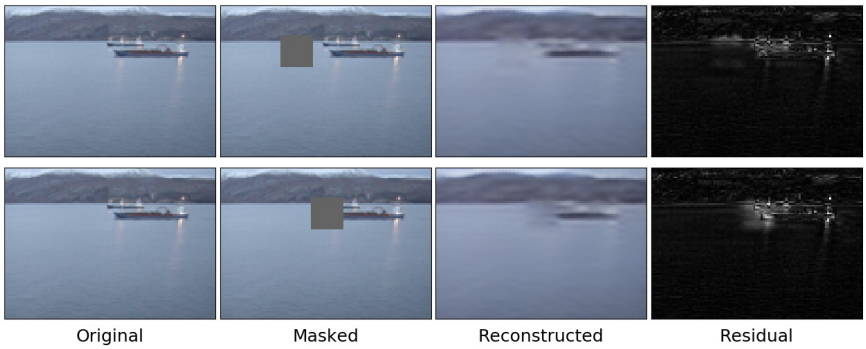


Figure 6.1: An image of two vessels are masked at different areas. The autoencoder reconstructs the images and inpaints the masked areas. **Top row**: The masked ocean is inpainted quite accurately, causing a low residual. **Bottom row**: The mask covers the bows of the vessels. The inpainting is not very accurate, causing a large residual.

After training, the model is applied to test images. Each test image $\mathbb{O}$ is masked at every possible cell in the grid, forming a minibatch $\mathbb{O}_m$ of $m \times n$ images. The model reconstructs $\mathbb{O}_m$ such that the masked areas are inpainted. The inpainted areas merged together to a single inpainting $\mathbb{I}$. The residual $\mathbb{R}$ is the absolute difference of the input image $\mathbb{O}$ and the inpainted output image $\mathbb{I}$, converted to single-channel grayscale representation. Areas are labeled as objects if the pixel value is greater than a given

threshold. The threshold is a tuning-parameter that is adjusted according to the user requirements. A lower threshold leads to more detections, both true and false positives, while a higher threshold leads to less detections. The object detection algorithm is summarized in Algorithm 1.

Algorithm 1: Object detection with inpainting autoencoder.

Set inpainting grid $m \times n$

for *training iterations* **do**

- Load minibatch $\mathbb{B}$ of training images
- Generate masked minibatch $\mathbb{B}_m$ by randomly apply one of $m \times n$ masks to each image in $\mathbb{B}$
- Train on $\mathbb{B}_m$ and update model parameters

end

Set threshold T

for *test iterations* **do**

- Load single test image $\mathbb{O}$
- Generate masked image batch $\mathbb{O}_m$ of size $m \times n$ by masking all possible grid cells in $\mathbb{O}$ once
- Predict on $\mathbb{O}_m$ and merge $m \times n$ inpainted areas to single image $\mathbb{I}$
- Compute residual image $\mathbb{R} = |\mathbb{O} - \mathbb{I}|$
- Label $\mathbb{R} > T$ pixel-wise as objects

end

6.3 Autoencoder model

Bhattad et. al. [5] solves the inpainting problem in a similar fashion as [29], state-of-the-art for large hole inpainting when proposed in 2016. As discussed in Chapter 4, improvements have been been shown in more recent work. The autoencoder model used in this thesis is based on the model for image inpainting proposed by Iizuka et. al. [16] and modified to suit the detection approach of Bhattad et. al. [5]. Elaborated in Chapter 4, the inpainting model of [16] consist of two parts: an autoencoder trained

to minimize the pixel wise reconstruction residual and a discriminator trained to minimize an adversarial loss [11]. The autoencoder make the inpainting consistent with the area of adjacent pixels, while the discriminator make the inpainting look more realistic and less blurry.

Following the anomaly detection approach by [5], only the autoencoder minimizing the reconstruction residual is used for object detection algorithm presented in this thesis, meaning that the discriminator is excluded. This is reasonable since the original purpose of the discriminator is to increase the sharpness and detail level of the inpainting, while this might not be beneficial when the goal is to detect objects. It is undoubtedly desirable with high quality inpainting for both tasks, however forcing the model to generate a detailed inpainted image may be detrimental. Recall that the detection approach is based on that true objects are detected if they stand out in the residual. If high-level details are generated in the inpainted image, they are likely to stand out in the residual in the same fashion as the true detailed objects, causing additional false positives. Only including the autoencoder also simplifies the model.

6.3.1 High level hyperparameters

Mask

The size of the masks that should be inpainted has a huge impact on the difficulty of the inpainting task, in addition to affect what objects are most likely to be detected. Clearly, larger masks are far more difficult than small ones and will be inpainted with lower quality. A model inpainting large masks will therefore require more training to achieve similar inpainting quality as a model inpainting smaller areas, if at all.

The mask size affect, but does not dictate, what objects can be detected depending on their size. Consider an example image showing a vessel surrounded by ocean. If a mask covers the entire vessel, the inpainting is likely to be ocean only, leading to a significant reconstruction residual. If a mask covers half of the vessel, the inpainting is likely to be consistent with the local pixels showing the other half of the vessel, leading to a less significant reconstruction residual.

A far less significant hyperparameter of the mask is the mask color. The approaches

in Chapter 4 varies from black [5], white [29] and the mean pixel value of the dataset [16]. In this work, the mean pixel value is used. The most likely implication of this choice is that the loss decreases slightly faster in the beginning of the training process.

Image size and computational load

The original dimension of the images is $1920 \times 2560 \times 3$, with the first two axes representing the spatial extent and the third axis representing RGB color channels. In comparison, the face images in the experiments of [5] are 300 times smaller, with the dimension $128 \times 128 \times 3$. Showed by [16], the relationship between image size and the computation time is approximately linear, given their full inpainting model. It is reasonable to assume a similar relationship also holds for the reduced model of inpainting autoencoder alone.

To reduce the computational load, the images are resized spatially to $144 \times 192 \times 3$ by bicubic interpolation. This is a quite aggressive down-sampling, leading to less detailed images. Even though discarding information is far from ideal, the it makes training on far more images possible within a reasonable amount of time. To be elaborated in Chapter 7.1.1, training on 1,723,299 resized images can be done in eight days on the available GPU. Additionally, training on full-resolution images would require the batch size being reduced to 1 or 2 images per batch to avoid running out of memory, reducing the speed-up effect of batch normalization [44].

6.3.2 Architecture

The architecture of the inpainting autoencoder of [16] is the given in Table 6.1. The input images are reshaped to have dimension height, width and channels $H \times W \times C = 144 \times 192 \times 3$. The encoder consist of convolutional layers, two of them with a stride $S = 2$, such that the bottleneck of the autoencoder has spatial dimensions $\frac{H}{4} \times \frac{W}{4}$. Four layers of dilated convolutions with increasing dilation rate increase the receptive field significantly. Recall that the receptive field corresponds to the spatial extent in the input image used to compute the activation of a neuron at a given layer, and needs to be larger than the masked area if it should be inpainted. The decoder consist of reg-

ular and transposed convolutional layers. The stride in the transposed convolutions up-samples the spatial dimension to equal the input dimension. Throughout the autoencoder, the number of filters doubles for each spatial down-sampling and is halved for each up-sampling. The ReLU activation function [27] with range $[0,\infty]$ and batch normalization [17] is used after each convolution except for the very last one, which has no batch normalization and Sigmoid as activation function. Batch normalization normalize and re-parametrize each batch of inputs along their spatial axes independently for each channel, speeding up training of deep neural networks significantly. Sigmoid has bounded output in the range $[0,1]$.

Following the model architecture, the selected training hyperparameters are also given by [16]. As loss function, the model is trained to minimize the pixel wise mean squared error (MSE). Discussed in Chapter 4.2.1, both mean absolute error (MAE) and MSE are common choices for training autoencoders to minimize the pixel wise difference. In comparison with MAE, MSE penalize large errors more and small errors less due to the squared norm. For training the inpainting autoencoder proposed by [29], both types of losses were tested without any significant differences.

Further following [16], to optimize the training process, AdaDelta [47] is selected. AdaDelta is based on stochastic gradient descent, applies momentum [31] and computes adaptive learning rates for the parameters individually. AdaDelta is slightly different from more recently proposed optimizers such as Adam [20] in the sense that it does not require the user to select a global learning rate.

Table 6.1: Autoencoder architecture.

Layer	Kernel size	Stride	Output dimension	Output channels	Dilation rate	Receptive field
Input	-	-	144 × 192	3	1	1
conv	(5,5)	(1,1)	144 × 192	64	1	5
conv	(3,3)	(2,2)	72 × 96	128	1	7
conv	(3,3)	(1,1)	72 × 96	128	1	11
conv	(3,3)	(2,2)	36 × 48	256	1	15
conv	(3,3)	(1,1)	36 × 48	256	1	23
conv	(3,3)	(1,1)	36 × 48	256	1	31
dil. conv	(3,3)	(1,1)	36 × 48	256	2	47
dil. conv	(3,3)	(1,1)	36 × 48	256	4	79
dil. conv	(3,3)	(1,1)	36 × 48	256	8	143
dil. conv	(3,3)	(1,1)	36 × 48	256	16	271
conv	(3,3)	(1,1)	36 × 48	256	1	279
conv	(3,3)	(1,1)	36 × 48	256	1	287
tran. conv	(4,4)	(2,2)	72 × 96	128	1	285
conv	(3,3)	(1,1)	72 × 96	128	1	293
tran. conv	(4,4)	(2,2)	144 × 192	64	1	291
conv	(3,3)	(1,1)	144 × 192	64	1	299
conv	(3,3)	(1,1)	144 × 192	3	1	307
Intermedialte layers	ReLU and batch norm. after each conv. layer					
Loss function	Mean Squared Error					
Optimizer	AdaDelta					

Chapter 7

Experiment and results

The object detection algorithm described in Chapter 6 is implemented and tested. The experiment involves training of the inpainting autoencoder, in addition to details about how the algorithm is evaluated. The evaluation results are then provided.

7.1 Experiment description

7.1.1 Autoencoder training

Selected training data

In general, a learning based model generalizes better when trained on more data [7]. Also, they learn faster if the examples they train on contain maximum information [28]. Described in detail in this section, several steps of selection are made that effectively reduce the amount of training data. This was found to be necessary to make training feasible within a reasonable amount of time, having a single GPU at disposal. The first step applies to both training and test data. Step 3 and 4 are only applied to training data and are based on the findings in Chapter 5.3.

1. Define a dataset as images captured every 2.5 second by camera 5 and 11.

2. Shuffle the dataset and put aside 10% for testing purposes.

3. From the remaining training data, remove images captured with speed less than 6 m/s.

4. From the remaining training data, remove half of images with mean illumination intensity less than 50.

Of the total number of timestamps, given in Table 5.2, every 2.5 timestamp in average is is selected to be used for training and testing. Images are acquired with a frame rate of 1 frame per second, resulting in that images used are captured every second and every third second, every other time. This exact selection is slightly odd, however the explanation is simply that the the model was first was trained on images captured every fifth second, before the size of the training dataset was doubled.

Further, a simplification is made by only including images captured by camera 5 and 11. Visualized in Figure 5.2, these images do not show any part of the ownship in the foreground, and can be used for training and testing directly. Images captured by the other cameras could also be used, but the pixels showing the ownship should have been removed due to the reason explained in the following paragraph.

Following the detection approach of [5], the autoencoder model should be trained on images showing as much ocean and as little objects as possible. This facilitates the fully trained model to reconstruct ocean more accurate than objects desirable to detect. Based on the data exploration in Chapter 5.3, training data should be selected carefully to ensure that it fulfills its purpose, motivating for step 3 and 4.

Given in step 3, images are filtered based on the speed of the ownship at the time the image is captured, as described in Chapter 5.3.1. All images captured when the speed is less than 6 m/s are removed. Visualized in Figure 5.4, this removes images captured when the ownship is docked, in addition to a small number of images taken with low speed nearby harbour. The speed filtering effectively reduces the number of training images with 21%.

Step 4, discussed in detail in Chapter 5.3.2, aims to balance the dataset between dark images captured at night and brighter images captured at day. Images captured at day contain far more information, as seen in Figure 5.3. Visualized in Figure 5.5 and Figure 5.6, randomly removing half of the images with mean illumination intensity below

50 yields a overweight of images captured in daylight without discarding dark images completely. 20% of the training images are removed by this process.

The entire process of selecting training data leads to a dataset of 1,206,031 timestamps. $\frac{1}{9}$ of the dataset is put aside for validation during training, leaving 1,072,028 timestamps for training. Due to the malfunction of camera 5 briefly mentioned in Chapter 5.1, a total of 1,723,299 images were used for training.

Having a large enough training dataset is a common topic of concern for training any neural network. In comparison, [5] train on 125,253 images in their anomalous face detection experiment, while [16], using the same inpainting autoencoder model as in this thesis, train on 8,097,967 diverse images originally meant for scene classification. In this thesis, the only relevant scene is the maritime environment seen from the ownship. Thus, the selected training dataset is likely to include enough images to make the autoencoder model generalize well.

Implementation details

Details of hardware and software used for implementation is given in Table 7.1.

Table 7.1: Software and hardware implementation details.

Hardware	Graphical processing unit (GPU)	Nvidia GTX 1080 Titan
	GPU memory	11 GB
Software	Operative System	Ubuntu 16.04
	Computing platform	CUDA 8.0
	Programming Language	Python 3.6
	Deep learning libraries	Keras and TensorFlow

Training process

Before training, the weights of the network are initialized randomly from a limited uniform distribution, while the bias terms are initialized to zero. For every eighth training batch during training, the model predicts on a single batch of validation data and computes the loss. The batch size is set to 16.

The training and validation loss throughout training is given in Figure 7.1. As expected, the losses drop quickly in the beginning of training. Further, the losses con-

Table 7.2: Details of the training process.

Images	1,723,299
Batch size	16
Batches	134,024
Training time	8 days

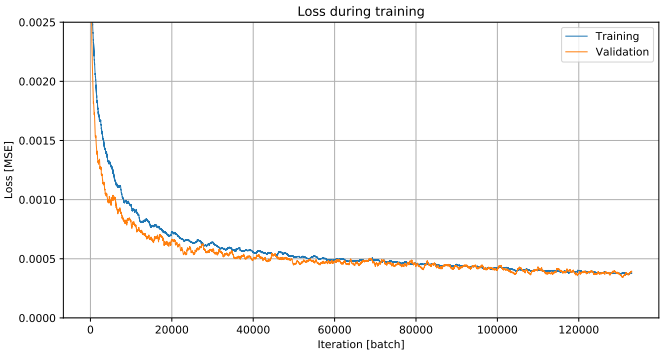
tinue to drop, but at a far lower rate, both approximately from 0.0006 to 0.0004 during the last 100,000 iterations. Based on the trend of the losses, it is likely that more training would lower the losses furthermore. Since the validation loss follows the training loss accurately, the model is not suspected to overfit to the training data to any noteworthy extent, additionally motivating for more training. However, for practical reasons the training is limited to train on all images in the described dataset once. The completed training process takes approximately eight days on a single GPU, implemented with the hardware and software given in Table 7.1.

Equally interesting as the losses, during and at the end of training, are the visual reconstructions, in particular the inpainted areas. Figure 7.2 shows four images from a validation minibatch, masked at different areas. Most of the masked areas are inpainted with approximately the same quality as the rest of the reconstructed image. However, the quality depends on what features the masks cover. This become more clear in Figure 7.3, in which the very left image from Figure 7.2 is masked and inpainted at all areas, yielding a fully inpainted image. The inpainted image illustrates that different image features are inpainted with different quality, creating a larger residual at the areas imaging shoreline. Still, the inpainted image is in general very blurry.

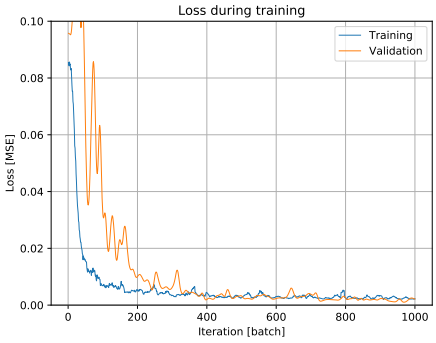
7.1.2 Evaluation

Test data selection

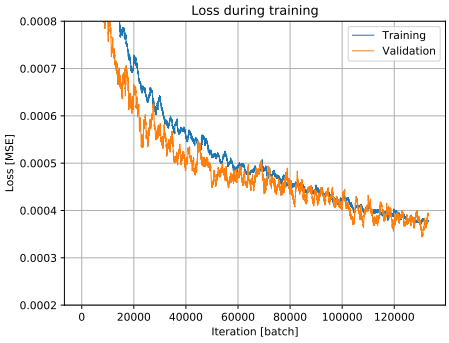
Before the detection algorithm is trained, 10% of the image data is randomly selected and put aside for testing, ensuring that there is no overlap in between training and test data. On average, the original image data do not contain a lot of objects. To test the algorithm properly, a diverse test dataset with as many objects as possible should be prepared. An efficient way to do this is to select data based on the presence of vessels



(a) The entire training process.



(b) The first 1000 iterations.



(c) The entire training process.

Figure 7.1: Training and validation loss during training.

signalling their nearby position by AIS. One should note that there is no guarantee that the a vessel is visible in the images even though its AIS signal indicates that it is nearby, but the probability increase significantly. Hence, the data put aside for testing is reduced to only contain images captured with one or more vessels within a range of one kilometer. Among the about 1,200 remaining images, a final subset X_{dn} of 203 images is selected manually to maximize the number of objects and diversity. X_{dn} contain two subsets: X_d consist of images captured in daylight, while X_n consist of images captured at night. Table 7.3 provide the number of images and objects in each of the test datasets.

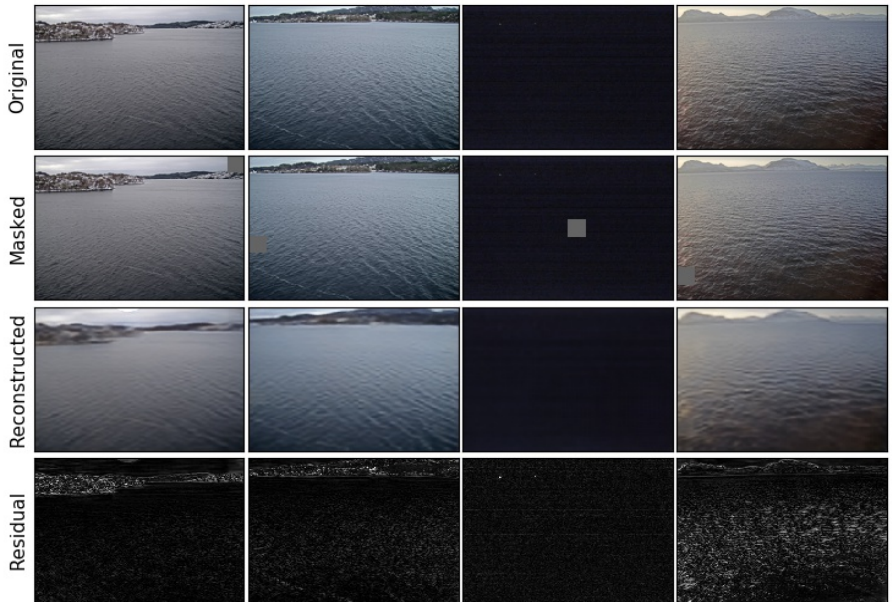


Figure 7.2: Validation images at the end of training. The autoencoder reconstructs the images and inpaints the masks with a fairly high quality. In the very right image, the mask covers reflections that are less accurately inpainted, hence a larger residual is obtained.

Figure 7.3: A validation image is masked and inpainted at all areas. The inpainted image is overall very blurry, although the blurriness is more explicit for the areas representing the shoreline than the ocean. The ocean is partially inpainted with a texture of horizontal elements.

Labeling of ground truth

The ground truth (GT) in the test data is labeled manually with the two classes given in Table 7.4. The object class is used to mark all areas containing an object the algorithm

Table 7.3: Test datasets.

Name	Description	Images	Objects
X_d	Day	147	225
X_n	Night	56	35
X_{dn}	$X_d \cup X_n$	203	260

should detect. The background class is used to mark the background areas that should be excluded from testing. This include any areas of the shore or the horizon and above. It does not make sense to test the object detection algorithm on the background since this is a completely different task than what the algorithm is designed for.

Ideally, the labeling should be on a per-pixel level, since the algorithm output per-pixel detections. Also, this would increase the accuracy of the testing. Yet, to reduce the manual workload, rectangular boxes are used to mark areas based on its content, tightly surrounding the objects that the algorithm should detect. In some cases, labeling is easy since the images clearly show what type of objects is present. In other cases, it is very difficult to be certain if the object is actually a vessel, a rock or even an islet. Due to the uncertainty, these are also labeled as objects, even though the algorithm is not designed to detect islets and shore.

The labeling is done by visual inspection of the images in full resolution. Examples of GT boxes are given in the visual results in Section 7.2.2.

Table 7.4: Labeled classes in test images.

Class	Description
Object	Vessels, boats, buoys, seamarks, rocks, sea farming equipment, unknown objects
Background	Shoreline or horizon and above

Performance metrics

Recall R and precision P are used to measure the performance of the detection algorithm, computed by the number of true positives tp, false negatives fn and false positives fp. False positives are also used directly as a performance metric.

Recall measure the number of true positives within the total number of positives.

The total number of positives include true positives and false negatives, i.e. recall is given by $R = \frac{tp}{tp+fn}$. The test data includes in total 260 GT boxes surrounding positives, as given in Table 7.3. A true positive is obtained if a single pixel within the GT box is predicted positive, while a false negative is obtained if no pixels within the GT box is predicted positive. This is summarized in Table 7.5.

Table 7.5: True positives and false negatives for recall computation.

tp	Number of GT boxes including at least a single predicted positive pixels.
fn	Number of GT boxes including no predicted positive pixels.

Precision is used to measure the number of true positives within the total number of predicted positives. The predicted positives include true and false positives, i.e. precision is given by $P = \frac{tp}{tp+fp}$. Whereas the total number of positives used for recall is given by the fixed number of objects, predicted positives used for precision varies depending on the threshold. At most, predicted positives can be all 4,915,200 pixel in the image.

Often, predicted positive pixels are adjacent to each other since they have detected the same object. Therefore, it makes sense to consider them as one, defined as a single predicted positive cluster. Clearly, there cannot be more predicted positives clusters than predicted positive pixels. Since predicted positives pixels and clusters provide different information, precision can be computed with both definitions, specified in Table 7.6. Predicted positive pixels that are orthogonal or diagonal adjacent to each other are considered a cluster.

Table 7.6: True positives and false positives for precision computation.

pixels	tp	Number of predicted positive pixels within the GT boxes.
	fp	Number of predicted positive pixels outside the GT boxes.
clusters	tp	Number of predicted positive clusters within the GT boxes.
	fp	Number of predicted positive clusters outside the GT boxes.

7.2 Results

The following sections show how the object detection algorithm perform on the two test subsets X_d and X_n, containing images captured in day and night, respectively. Also, the

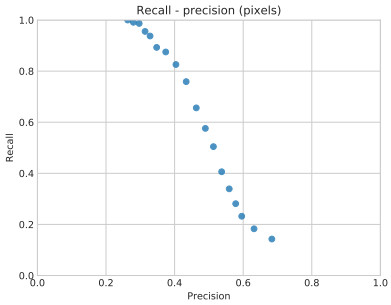
results on the combined test set X_{dn} are provided. In Section 7.2.2, visual results are given.

For each of the test sets, recall is plotted against precision and against the number of false positives. The results are obtained by varying the threshold, as described in Algorithm 1. Precision and false positives are computed both by individual pixels and clusters of pixels. The number of false positives are given as mean per image.

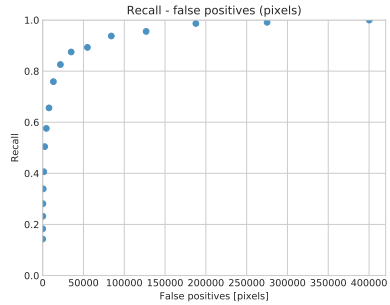
7.2.1 Quantitative results

Day

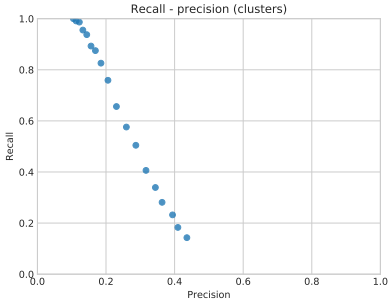
The performance on X_d is given in Figure 7.4. Computed by pixels, given in Figure 7.4a and Figure 7.4b, a perfect recall $R = 1.0$ is given for precision $P = 0.3$ and false positives $fp \approx 400,000$, which is as much as 8% of the entire image. Lowering the recall to $R = 0.9$ yields far less false positives, $fp \approx 60,000$, but precision does not increase by a lot. Computed by clusters, given in Figure 7.4c and Figure 7.4d, a perfect recall $R = 1.0$ is given for precision $P = 0.1$ and false positives $fp \approx 22,500$. False positives are reduced to $fp \approx 7,000$ for a recall $R = 0.9$. Overall, precision is low for both pixel and cluster detections.



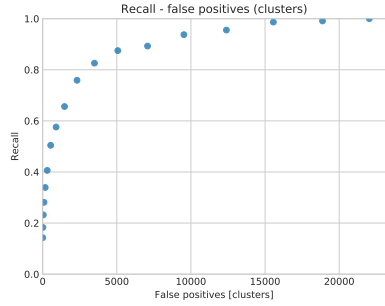
(a) Recall vs. precision (pixels).



(b) Recall vs. false positives (pixels).



(c) Recall vs. precision (clusters).



(d) Recall vs. false positives (clusters).

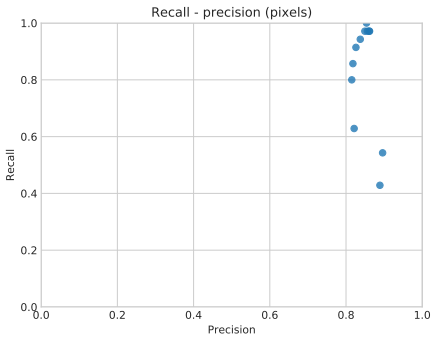
Figure 7.4: Results on test data X_d captured at day.

Night

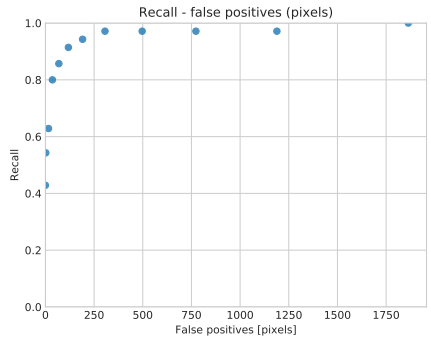
The performance on X_n is given in Figure 7.5. Computed by pixels, given in Figure 7.5a and Figure 7.5b, a perfect recall $R = 1.0$ is given for precision $P = 0.85$ and false positives $fp \approx 1,800$.

Computed by clusters shown in Figure 7.5c and Figure 7.5d, a perfect recall is given for precision $P = 0.7$ and false positives $fp \approx 120$. If a low recall is acceptable, $R = 0.4$, false positives can be reduced to $fp \leq 1$ per image.

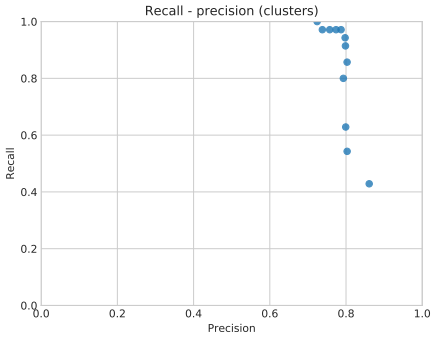
For both pixels and clusters, precision is high, i.e. $P \geq 0.7$ for all recalls.



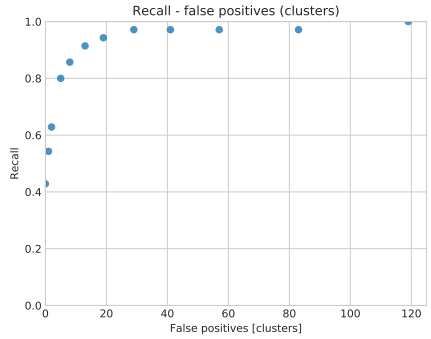
(a) Recall vs. precision (pixels).



(b) Recall vs. false positives (pixels).



(c) Recall vs. precision (clusters).

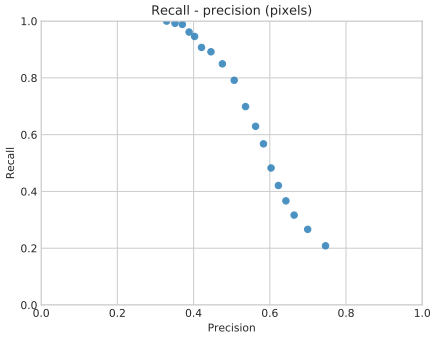


(d) Recall vs. false positives (clusters).

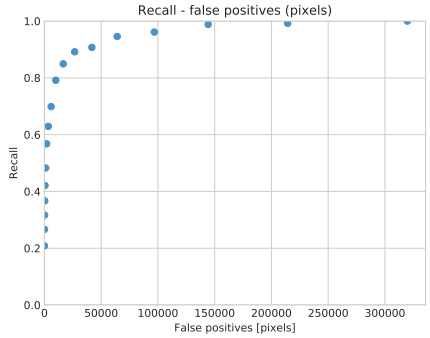
Figure 7.5: Results on test data X_d captured at night.

Day and night

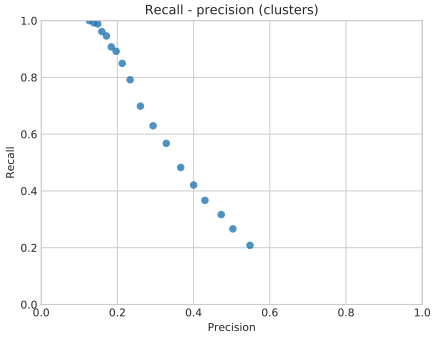
The performance on the combined dataset X_{dn} is given in Figure 7.6. Due to the larger number of images in X_d than in X_n, the combined performance show similar patterns as the performance of X_d alone.



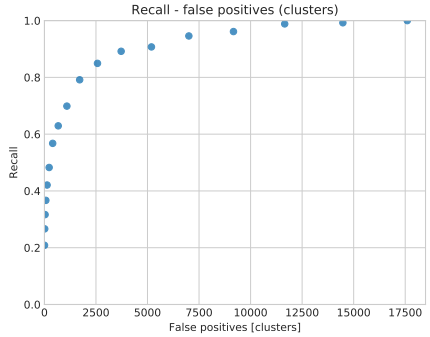
(a) Recall vs. precision (pixels).



(b) Recall vs. false positives (pixels).



(c) Recall vs. precision (clusters).



(d) Recall vs. false positives (clusters).

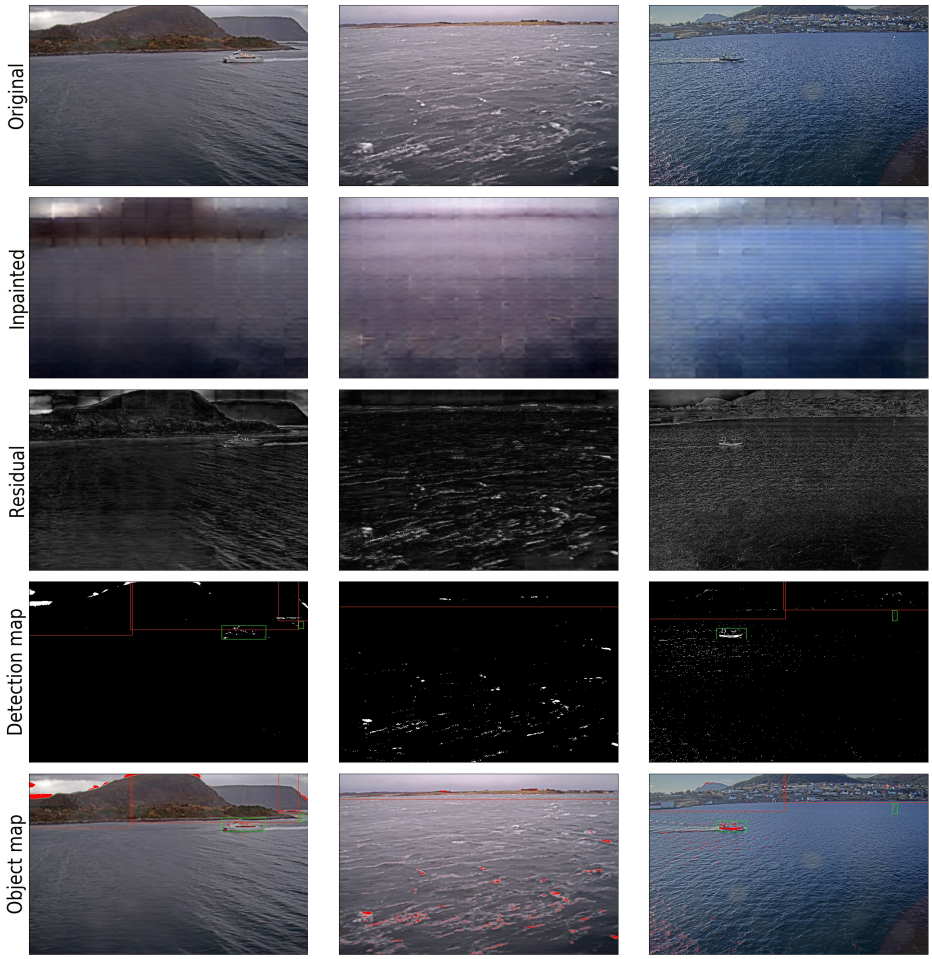
Figure 7.6: Results on test data X_{dn} captured at day and night.

7.2.2 Visual results

The following figures show the work flow and the output of the detection algorithm applied to test data. The figures are organized as follows: the first row show original test images; the second show inpainted images by the inpainting autoencoder; the third row show the residual of original and inpainted images; the fourth row show the detections

and ground truth (GT) boxes; the fifth row show the detections and GT boxes on top of the original image. The threshold for detection are in following figures set to $T = 130$, which is the threshold used to obtain a recall $R = 0.9$ on the combined test set X_{dn}. In the detection and object map, green GT boxes indicates objects and orange GT boxes indicates background.

A PDF-viewer with zoom is recommended for studying the visual results. Perhaps most interesting is the residual image, since the detection map depends on the set threshold. Objects can only be detected if they "stand out" visually, i.e. their pixel values are greater than the surroundings.

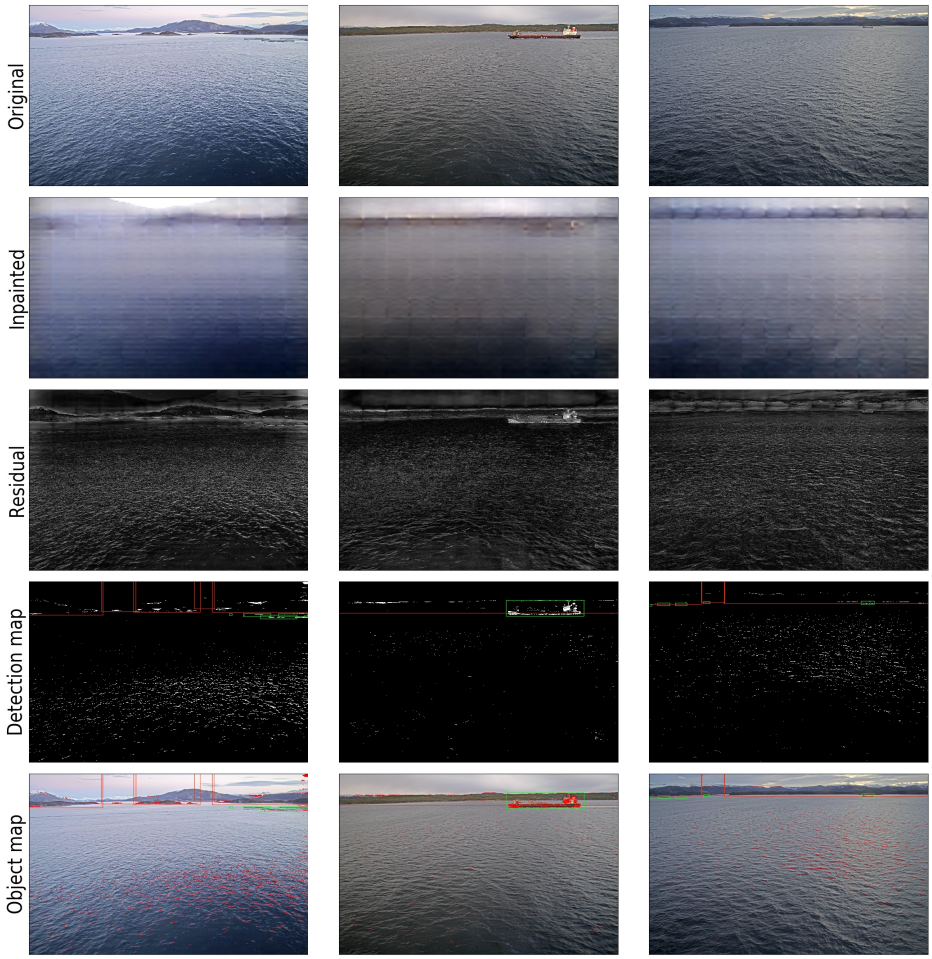


(a) The vessel is detected. The seamark to the right close to shore is in the residual darker than the surroundings, i.e. it cannot be detected no matter the threshold.

(b) Sea spray is detected.

(c) Both the vessel and the seamark to the right are detected. Reflections and waves are detected.

Figure 7.7: Test images from X_d.

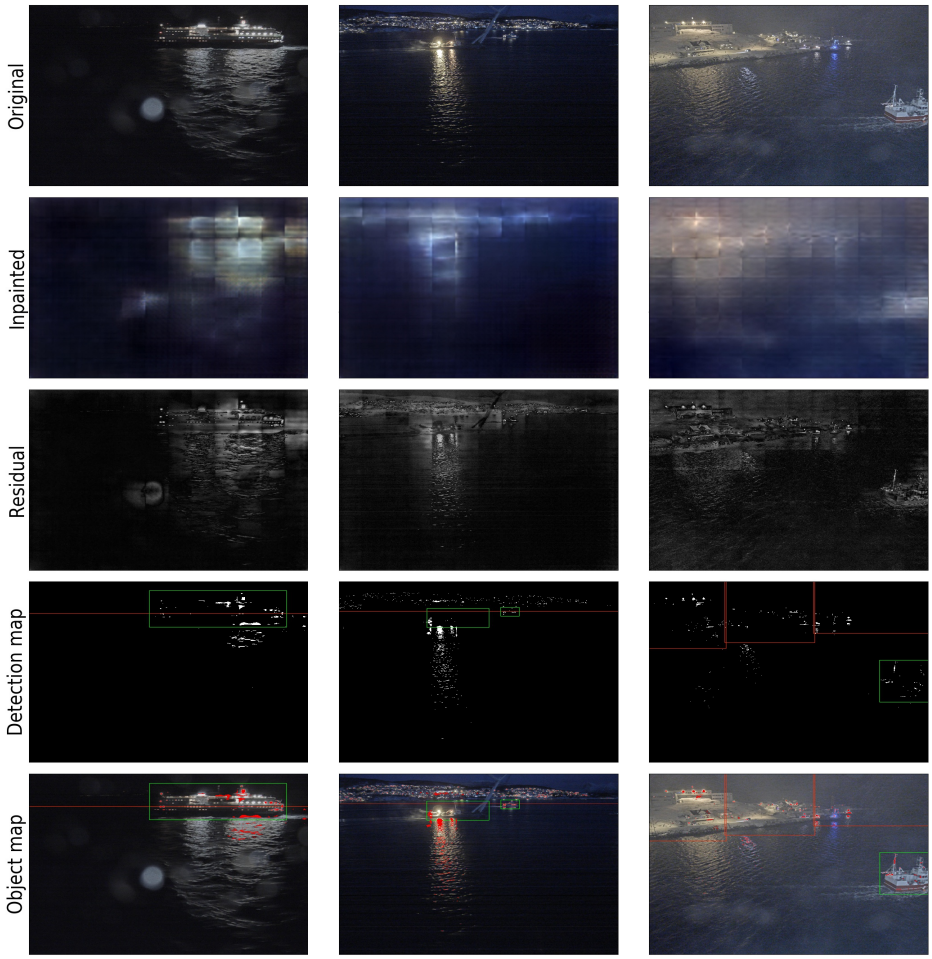


(a) Sea farming cages to the right and small island are detected. Waves are detected. Also, note the large white spot in the top of the inpainting.

(b) The ship is clearly detected. Waves are detected.

(c) To the right, the vessel is detected. To the left, a vessel and sea farming cages are detected for a lower threshold. Waves are detected.

Figure 7.8: Test images from X_d.



(a) The lights from the ship and the reflections in the ocean are detected.

(b) The stern of the left vessel and reflections in front are detected, however its brightest light source requires lower threshold to be detected.

(c) Various parts of the vessel and reflections are detected

Figure 7.9: Test images from X_n.

7.2.3 Results analysis

The visual results show that objects in most cases are covered in the inpainted image, yielding a larger residual for objects than for other areas. This make them possible to detect for the correct threshold. However, this is equally true for ocean features such as waves, sea spray and reflections, leading to a high number of false positives. In X_n, the false positives typically originates from reflections of light sources, located quite close to the true objects. In X_d, false positives are spread out all over the ocean surface in a high number.

The high number of false positives is even more evident in the results for X_d given in Section 7.2.1. For clustered detections, precision is low, meaning that on average there are more false positives than true positives. For X_n, precision is high, accordingly most positives are true.

An important aspect about the quantitative results is that they are based on a quite small test dataset, lowering the accuracy of the computed metrics. This is in particular true for the night set X_n of only 56 images and 35 objects. For this set, the recall-precision curves in Figure 7.5 seem slightly noisy, while the recall-false positive curves are more distinct.

7.3 Discussion

An important observation in the visual results is that the size of masks that have been inpainted the objects location relative to the grid of inpaintings have a huge impact on value of the residual. As pointed out in Chapter 6, larger masks are more difficult to inpaint with high quality since the areas in the middle of the mask is located further away from the non-masked areas. Therefore, objects are more likely to be detected if they are covered by the middle of a mask, since this cause a large residual. The opposite is the case in Figure 7.9b, where a light source is located at the edge of two masks, causing a small residual and no detection for the set threshold. Additionally, the fact that the mask size is a set as a fixed hyperparameter makes the algorithm scale variant, i.e. likely to detect certain sized objects more frequent than others. This could be mitigated

by training the model to inpaint masks of various sizes.

The autoencoder model would benefit from more training, since the loss continues to drop and the reconstructions still are very blurry, in particular the inpainted part of the reconstruction. In addition, some of the inpainted images, e.g. Figure 7.8a, show large white spots, a sign of need for more training. Another option to more training is to modify the training scheme to speed up the learning process. In the experiment presented in this thesis, the autoencoder model is trained to minimize the residual of the original and the reconstructed image in the same way as in [5]. However, the detection algorithm does only make use of the inpainted part of the reconstructed image. A more efficient training scheme would be to minimize the residual of the original and the inpainted image directly. This could be achieved by put all weight on the inpainted area of the reconstruction in the loss, and discard the non-inpainted part of the reconstruction. The final speed-up effect of this modification is very difficult to predict. The reconstruction problem is removed, but the inpainting problem remains equally difficult.

The autoencoder was trained on images carefully selected to contain mostly ocean features. The idea was that this should make the autoencoder inpaint features such as waves and sea spray more accurately than object features. When inspecting the results, it is difficult to argue whether this really was achieved or not. Seemingly, the ocean parts of the inpainted images look more realistic than the background and objects, typically showing a texture of horizontal elements, but they are still blurred. The inpainted ocean might look more realistic just because the ocean surface is smoother and less detailed than objects in the first place. An additional observation about the training data selection, is that one cannot control for the exact content of all images following an unsupervised regime. Based on all images shown in this thesis, one might suspect waves to be visible in a large portion of the images, while sea spray is visible less frequent.

The algorithm is designed to detect objects on a per pixel-level, which is the same as binary classification of each individual pixel. This design has both benefits and limitations that is confirmed by the results. A benefit is that very small objects can be detected, at its smallest down to the size of covering only a single pixel. This allows for very

fine-grained detection. However, objects often cover a large number of pixels, resulting in that the algorithm detects the same object at several locations. Although it would be useful, the algorithm has no chance of knowing that it has detected a single object at several locations, rather than several objects located close to each other. Clustering adjacent pixel detections mitigates, but does not cure this problem.

Chapter 8

Concluding remarks

8.1 Conclusion

An unsupervised object detection algorithm for images is implemented and tested. The algorithm is based on recent research within anomaly detection [5], and is build upon an autoencoder that inpaints masked parts of images. The output of the autoencoder highlights objects which enables them to be detected. The detection algorithm relies on the assumption that objects are represented as anomalies in the image data.

The results show that the detection algorithm is capable of detecting all objects, although it also detect features not considered to be objects, such as waves. This result is expected, due to the briefly mentioned assumption of the detection algorithm. Consequently, a high rate of false positives is obtained. If 90% of the objects should be detected, i.e. a recall $R = 0.9$, the algorithm output 5,000 false positive clustered detections on average in each image, corresponding to a precision $P = 0.18$. If no more than a recall $R = 0.5$ is required, the false positives are reduced to 250 false positive clustered detections in each image.

The algorithm is tested in both daylight and in darkness. Due to the fact that objects in the test set are equipped with light sources, they are detected in both illumination conditions. In darkness, the algorithm output far less false positives since no ocean features are visible.

The autoencoder is trained on images of ocean and becomes able to inpaint masked areas, although with distinct blur. More training, different hyperparameter settings and a more efficient training scheme are likely to increase the inpainting quality, which again should contribute to higher detection rate.

One should note that evaluation is done on a relatively small test set. Hence, further testing in various conditions should be conducted to verify the results. Also, the obtained results should be compared with different types of detection algorithms. The detection algorithm may take part in a larger sensor fusion system if the overall system is able to handle the high number of false positives.

8.2 Further work

Extensive testing and comparison with existing methods are suggested for further work. In addition, several modifications and improvements of the detection algorithm should be explored. The following list summarizes the suggestions:

- Perform testing of the detection algorithm on a larger test set to increase the evaluation accuracy.
- Compare results to non-learning feature-based detection methods [25, 3] and off-the-shelf general-purpose supervised learning detection methods [24, 34].
- Train autoencoder on varying mask sizes to avoid inpainting grid pattern and make the detection algorithm more robust towards scale variance.
- Train autoencoder more and eventually on full resolution images.
- Develop the threshold for abnormality from being fixed to being based on a moving average [15].
- Incorporate the detection algorithm as part of a sensor fusion system.

Bibliography

[1] AIS transponders - Regulations for carriage of AIS. International Maritime Organization http://www.imo.org/en/OurWork/safety/navigation/pages/ais.aspx, 2018.

[2] M. Z. Alom, T. M. Taha, C. Yakopcic, S. Westberg, M. Hasan, B. C. V. Esesn, A. A. S. Awwal, and V. K. Asari. The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches. *CoRR*, abs/1803.01164, 2018.

[3] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. Speeded-Up Robust Features (SURF). *Comput. Vis. Image Underst.*, 110(3):346–359, June 2008.

[4] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester. Image inpainting. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 417–424. ACM Press/Addison-Wesley Publishing Co., 2000.

[5] A. Bhattad, J. Rock, and D. A. Forsyth. Detecting anomalous faces with 'no peeking' autoencoders. *CoRR*, abs/1802.05798, 2018.

[6] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3):15:1–15:58, July 2009.

[7] F. Chollet. *Deep Learning with Python*. Manning Publications, Nov. 2017.

[8] J. Dong, X. Mao, C. Shen, and Y. Yang. Unsupervised feature learning with symmetrically connected convolutional denoising auto-encoders. *CoRR*, abs/1611.09119, 2016.

[9] D. A. Forsyth and J. Ponce. *Computer Vision: A Modern Approach*. Prentice Hall Professional Technical Reference, second edition, 2002.

[10] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[11] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks. *arXiv*, 2014.

[12] H. He and E. Garcia. Learning from Imbalanced Data. *Knowledge and Data Engineering, IEEE Transactions*, 21:1263 – 1284, 10 2009.

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

[14] V. J. Hodge and J. Austin. A survey of outlier detection methodologies. *Artificial Intelligence Review*, 22(2):85–126, Oct 2004.

[15] L. Hou, V. Nguyen, D. Samaras, T. M. Kurc, Y. Gao, T. Zhao, and J. H. Saltz. Sparse autoencoder for unsupervised nucleus detection and representation in histopathology images. *arXiv*, 2017.

[16] S. Iizuka, E. Simo-Serra, and H. Ishikawa. Globally and locally consistent image completion. *ACM Trans. Graph.*, 36(4):107:1–107:14, July 2017.

[17] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[18] S. D. Jain, B. Xiong, and K. Grauman. Pixel objectness. *arXiv*, 2017.

[19] E. Jokioinen, J. Poikonen, M. Hyvönen, A. Kolu, T. Jokela, J. Tissari, A. Paasio, H. Ringbom, F. Collin, M. Viljanen, R. Jalonen, R. Tuominen, M. Wahlström, J. Saarni, S. Nordberg-Davies, and H. Makkonen. Remote and autonomous ships: The next steps. Technical report, Advanced Autonomous Waterborne Applications, Rolls Royce Marine, 2016.

[20] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[22] K. K. Krossholm. Unsupervised object detection in video from maritime environments. Fall project, Norwegian University of Science and Technology, Dec. 2017.

[23] Y. Lecun, L. Bottou, G. B. Orr, and K.-R. Müller. Neural networks: Tricks of the trade. In G. B. Orr and K.-R. Müller, editors, *Efficient BackProp*, pages 9–50. Springer, 1998.

[24] T. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. *CoRR*, abs/1708.02002, 2017.

[25] D. G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *Int. J. Comput. Vision*, 60(2):91–110, Nov. 2004.

[26] X. Mao, C. Shen, and Y. Yang. Image restoration using convolutional auto-encoders with symmetric skip connections. *CoRR*, abs/1606.08921, 2016.

[27] V. Nair and G. E. Hinton. *Rectified Linear Units Improve Restricted Boltzmann Machines*, pages 807–814. ICML'10. Omnipress, USA, 2010.

[28] G. B. Orr and K.-R. Mueller, editors. *Neural Networks : Tricks of the Trade*, volume 1524 of *Lecture Notes in Computer Science*. Springer, 1998.

[29] D. Pathak, P. Krähenbühl, J. Donahue, T. Darrell, and A. A. Efros. Context Encoders: Feature Learning by Inpainting. *CoRR*, abs/1604.07379, 2016.

[30] M. A. F. Pimentel, D. A. Clifton, L. Clifton, and L. Tarassenko. A review of novelty detection. *Signal Processing*, 2014.

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

[32] D. M. W. Powers. Evaluation: From precision, recall and f-factor to roc, informedness, markedness & correlation. Technical report, School of Informatics and Engineering - Flinders University, 2007.

[33] J. Redmon and A. Farhadi. Yolo9000: Better, faster, stronger. *arXiv*, 2016.

[34] J. Redmon and A. Farhadi. YOLOv3. Technical report, University of Washington, 2018.

[35] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[36] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. http://www.image-net.org/.

[37] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and F. Li. Imagenet large scale visual recognition challenge. *CoRR*, abs/1409.0575, 2014.

[38] R. Sceliski. *Computer Vision: Algorithms and Applications*. Springer, 2010.

[39] E. Shelhamer, J. Long, and T. Darrell. Fully convolutional networks for semantic segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(4):640–651, April 2017.

[40] 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*, 15:1929–1958, 2014.

[41] E. J. Tangstad. Visual detection of maritime vessels. Master's thesis, Norwegian University of Science and Technology, June 2017.

[42] UN-Business. International Maritime Organization. https://business.un.org/en/entities/13.

[43] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, pages I–511–I–518 vol.1, 2001.

[44] Y. Wu and K. He. Group normalization. *arXiv*, Mar. 2018.

[45] F. Yu and V. Koltun. Multi-scale context aggregation by dilated convolutions. *CoRR*, abs/1511.07122, 2015.

[46] J. Yu, Z. Lin, J. Yang, X. Shen, X. Lu, and T. S. Huang. Generative image inpainting with contextual attention. *CoRR*, abs/1801.07892, 2018.

[47] M. D. Zeiler. ADADELTA: an adaptive learning rate method. *arXiv*, abs/1212.5701, 2012.

[48] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus. Deconvolutional networks. In *In CVPR*, 2010.

[49] P. Zhu, H. Wang, T. Bolukbasi, and V. Saligrama. Zero-shot detection. *CoRR*, abs/1803.07113, 2018.