Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Marine Technology

Kasper W. Breistein

Applying Machine Learning Using Custom Trained Convolutional Neural Networks on Subsea Object Detection and Classification

Master's thesis in Marine Technology
Supervisor: Ingrid Schjølberg
June 2019

NTNU
Norwegian University of
Science and Technology

Kasper W. Breistein

Applying Machine Learning Using Custom Trained Convolutional Neural Networks on Subsea Object Detection and Classification

Master's thesis in Marine Technology
Supervisor: Ingrid Schjølberg
June 2019

Norwegian University of Science and Technology
Faculty of Engineering
Department of Marine Technology

NTNU
Norwegian University of
Science and Technology

NTNU

Norwegian University of Science and Technology

Master's Thesis in Marine Technology, Spring 2019

Kasper Endre Westbye Breistein

Applying Machine Learning Using Custom Trained Convolutional Neural Networks on Subsea Object Detection and Classification

Correct detection and classification of objects underwater is imperative for autonomous subsea intervention missions. However, detection based on classical computer vision is difficult and error-prone, due to the requirement of manually created feature extraction. This project seeks to investigate deep learning methods as an alternative to manual feature extraction for underwater objects. In addition to examine the effects of complementary training on images from above water, with background in the fact that above water images are easier, cheaper and more plentiful to come by in most circumstances.

Tasks:

1. Literature review. Investigate object detection methods.
2. Set up a lab environment. Develop picture database.
3. Implement and test CNN based method for detection.
4. Verify with drone operation using drone camera.
5. Write report

Supervisor:	Ingrid Schjølberg	**Start date:**	January 15th, 2019
Co-supervisor:	Mikkel C. Nielsen	**Due date:**	June 11th, 2019

Preface

This thesis is the final product of a *Master of Science* degree in Marine Technology with Marine Cybernetics specialization at the Norwegian University of Science and Technology (NTNU). The research period spans from January to June 2019, and is a continuation of the Project Thesis written in the fall of 2018.

The main motivation of this work is to investigate the possibilities for Deep Learning as a basis in designing object detection for Autonomous Underwater Vehicles (AUVs). This approach has the potential to severely reduce costs and increase subsea operability.

The project thesis assumes the reader has a basic understanding of cybernetics, subsea marine operations, machine learning and neural networks.

Kasper Endre Westbye Breistein

Trondheim, June 11, 2019

Acknowledgement

I would like to thank my project supervisor Professor Ingrid Schjølberg for her optimism and assistance during the semester. Additionally I am grateful to my co-supervisor Postdoctoral Fellow Mikkel Cornelius Nielsen for being a valuable discussion partner on the different aspects of neural networks. I also want to extend my kindest regards to family and friends who encouraged me during my years of study.

Kasper Breistein

Abstract

Due to aging equipment on the Norwegian continental shelf, the need for Inspections, Maintenance and Repair (IMR) operations are expected to increase considerably in the coming years. Remotely Operated Vehicles (ROVs) play a significant role in these operations today, but are expensive due to the accompanying ship with its crew. A solution which would immensely reduce operational costs are Autonomous Underwater Vehicles (AUVs). They would give the ability to monitor large areas, offer around-the-clock supervision and allow for instant access to assistance where needed. These AUVs require real-time decision making, where correct detection and classification of objects underwater is imperative. Detection based on classical computer vision is difficult and error-prone due to manually created feature extractions. A robust alternative might be deep learning methods, an area which has seen rapid advances lately. Thus the focus of this thesis, is how to detect and classify objects below water by using deep learning, and to examine the effects of training on complementary images from above water. The objective is to implement a deep learning object detector, train it on different image datasets, and evaluate the results on a fixed validation dataset. This is exemplified through training on images of a hammer, with the assumption that detection of any other object is similar in scope, just with different training data.

The Convolutional Neural Network (CNN) algorithm You Only Look Once version 2 (YOLOv2) was implemented in Python and trained on three different image datasets: above water, below water and hybrid (a combination of the other two). These datasets were collected from the Marine Cybernetics lab at Tyholt NTNU, using a BlueROV2 and its camera. Evaluation occurred on a dataset comprising of both above and below water images, totaling 9.2% of all the images collected. This resulted in Average Precision scores for each detector of 36%, 77%, and 92%, respectively. The results are as expected and indicate that object detection using deep learning is indeed possible. Higher precision was achieved with the hybrid detector, which was trained on above water images that are typically cheaper to produce.

Sammendrag

Med bakgrunn i aldrende utstyr på norsk kontinentalsokkel er behovet for inspeksjoner, vedlikehold og reparasjoner forventet å øke betydelig de kommende årene. Fjernstyrte undervannsfarkoster (ROV) spiller en vesentlig rolle i slike operasjoner i dag, men er svært dyre på grunn av det medfølgende skip og sitt mannskap. En løsning som ville redusere operasjonskostnadene betraktelig er autonome undervannsfarkoster (AUV). De ville gi muligheten til å monitorere store områder, tilby tilsyn døgnet rundt og muliggjøre umiddelbar tilgang på assistanse. Disse AUVene krever beslutningstaking i sanntid, der korrekt detektering og klassifisering av undervannsobjekter er avgjørende. Detektering basert på tradisjonelt maskinsyn er vanskelig og feilutsatt på grunn av behovet for å trekke ut karakteristikker manuelt. Et robust alternativ kan være maskinlæringsmetoder, et område som har sett kraftig forbedring i det siste. Dermed er fokuset i denne tesen, hvordan man kan detektere og klassifisere undervannsobjekter ved å bruke maskinlæring, samt å undersøke effekten det har å trene på komplementære overvannsbilder. Formålet er å implementere en objektdetektor basert på maskinlæring, trene den på forskjellige bildedatasett, og evaluere på et fast valideringsdatasett. Dette er eksemplifisert ved å trene på bilder av en hammer, med antagelsen om at detektering av andre objekter har lik metode, bare med forskjellig treningsdata.

Det nevrale nettverket (CNN) You Only Look Once versjon 2 (YOLOv2) ble implementert i Python og trent på tre forskjellige datasett: overvann, undervann og hybrid (en kombinasjon av de andre to). Disse datasettene ble innsamlet fra Marin Kybernetikk-laben på Tyholt NTNU, gjennom en BlueROV2 og dets kamera.
Evaluering ble gjort på et datasett som inneholdt både over- og undervannsbilder, og som totalt bestod av 9.2% av de innsamlede bildene. Dette resulterte i gjennomsnittlig presisjon (AP) for hver detektor på henholdsvis 36%, 77%, og 92%. Resultatene er som forventet og indikerer at deteksjon av objekter ved bruk av maskinlæring er mulig. Høyere presisjon ble oppnådd med hybriddetektoren, som var trent på overvannsbilder som generelt er billigere å produsere.

Contents

Preface i

Acknowledgement ii

Abstract iii

Sammendrag iv

List of Figures vii

List of Tables ix

Abbreviations xii

1 Introduction 1
- 1.1 Background and Motivation . . . 1
- 1.2 Object Detection . . . 2
- 1.3 Earlier Work . . . 3
  - 1.3.1 Viola-Jones Algorithm . . . 3
  - 1.3.2 Histogram of Oriented Gradients . . . 4
  - 1.3.3 The Deep Learning Revolution . . . 5
  - 1.3.4 R-CNN . . . 6
  - 1.3.5 YOLO . . . 7
  - 1.3.6 Underwater Visual Tracking . . . 8
- 1.4 Thesis Objective . . . 9
- 1.5 Thesis Structure . . . 10

2 Theory 11
- 2.1 Deep Learning . . . 11
  - 2.1.1 Supervised Learning . . . 11
- 2.2 Deep Feedforward Networks . . . 12
- 2.3 Activation Function . . . 13
  - 2.3.1 Rectified Linear Unit . . . 14
  - 2.3.2 Leaky Rectified Linear Unit . . . 14
  - 2.3.3 Sigmoid Activation Function . . . 15
  - 2.3.4 Tanh Activation Function . . . 15
- 2.4 Output Function . . . 15

2.5 Optimizing With Gradient Descent . . . 16
2.6 Convolutional Neural Networks . . . 17
2.6.1 Convolution . . . 17
2.6.2 Sparse Interactions . . . 18
2.6.3 Parameter Sharing . . . 19
2.6.4 Equivariant Representations . . . 19
2.6.5 Pooling and Downsampling . . . 20
2.7 Evaluation Metrics . . . 20
2.7.1 Intersection Over Union . . . 21
2.7.2 Precision x Recall Curve . . . 22
2.7.3 Average Precision . . . 23
2.8 YOLO . . . 23
2.8.1 Prediction Tensor . . . 24
2.8.2 Loss Function . . . 25

3 Data Collection **29**
3.1 Object . . . 29
3.2 Camera . . . 30
3.3 ROV Setup . . . 30
3.4 Above Water Data Generation Setup . . . 31
3.5 Below Water Data Generation Setup . . . 32
3.6 Validation Dataset . . . 34
3.7 Labels . . . 34

4 Implementation of CNN Method **35**
4.1 Python . . . 35
4.2 Google Colaboratory . . . 35
4.3 Tensorflow . . . 36
4.4 Darkflow . . . 37
4.5 Building the Neural Network . . . 37
4.6 Training the Neural Network . . . 38
4.7 Validating the Neural Network . . . 38

5 Results **39**
5.1 Time Spent Manually Labelling Images . . . 39
5.1.1 Above Water Dataset . . . 39
5.1.2 Below Water Dataset . . . 40
5.1.3 Hybrid Dataset . . . 41
5.2 Object Detector Validation . . . 41
5.2.1 Object Detector Trained on Above Water Dataset . . . 42
5.2.2 Object Detector Trained on Below Water Dataset . . . 43
5.2.3 Object Detector Trained on Hybrid Dataset . . . 44
5.3 Example Images . . . 44
5.3.1 Similar Images Yield Different Results . . . 45
5.3.2 IOU Too Strict . . . 48
5.3.3 Images Only Found by Lower Detectors . . . 49
5.3.4 Difficult Images . . . 52

6 Discussion **53**
6.1 Time Spent Manually Labelling Images 53
6.2 Object Detector Validation 53
6.3 Similar Images Yield Different Results 54
6.4 IOU Too Strict 54
6.5 Images Only Found by Lower Detectors 55

7 Conclusions and Further Work **57**
7.1 Conclusions 57
7.2 Further Work 58

A Label Output Format **59**

Bibliography **60**

List of Figures

1.1 Autonomous Underwater Vehicles Performing Inspection[1] 1
1.2 Remotely Operated Vehicle Working on a Subsea Installation[3] 2
1.3 Main Features of the Viola-Jones Algorithm[8] 3
1.4 Representation of a Face Using Histogram of Oriented Gradients[9] 4
1.5 Test Images with Labels and Probability Distribution From Krizhevsky et al.[11] 5
1.6 System Overview of R-CNN Algorithm[13] 6
1.7 Object Detection Samples from YOLOv2[18] 7
1.8 Hexapod Underwater Visual Target Tracking Using YOLOv2[21] 8
1.9 CNN Object Detector Flow Chart 9

2.1 Hierarchical Structure of Artificial Intelligence, Machine Learning and Deep Learning[22] 12
2.2 Supervised Learning Overview[22] 12
2.3 A Fully Connected Feed Forward Network[23] 13
2.4 Rectified Linear Unit Activation Function[24] 14
2.5 Leaky Rectified Linear Unit Activation Function[24] 14
2.6 Sigmoid Activation Function[24] 15
2.7 Tanh Activation Function[24] 15
2.8 Gradient Descent Illustrated as a Ball Seeking Lowest Position of a Path[28] 16
2.9 Typical Stages of a Convolutional Neural Network Layer[24] 17
2.10 Convolution as the Dot Product Between Kernel and Input[22] 18
2.11 Convolution Using an Edge Detection Kernel[29] 18
2.12 Sparse Interactions Between Nodes[24] 19
2.13 Parameter Sharing Between Nodes[24] 19
2.14 Max Pool and Downsampling Between Nodes[24] 20
2.15 How Pooling Causes Rotation Invariance[24] 20
2.16 Intersection Over Union[30] 21
2.17 Precision x Recall Curve[30] 22
2.18 Average Precision from Integrating the Precision x Recall Curve[30] . . . 23
2.19 Architecture of Darknet-19[18] 24
2.20 System Overview of YOLO Algorithm[17] 25

3.1 Tool Used In This Report[32] 29
3.2 Raspberry Pi Camera Module V2 with Wide Angle Lens[34] 30
3.3 BlueROV2 Heavy Configuration[35] 30
3.4 Single Hammer Setup 31
3.5 Dual Hammer Setup 32

3.6 ROV Setup in Pool 33
3.7 Underwater Images Taken From The ROV 33
3.8 LabelImg Software[36] 34

4.1 Tesla K80 Graphics Card[39] 36
4.2 TensorFlow Machine Learning Library Logo[40] 37

5.1 Precision x Recall Curve and Average Precision for Above Water Dataset 42
5.2 Precision x Recall Curve and Average Precision for Below Water Dataset 43
5.3 Precision x Recall Curve and Average Precision for Hybrid Dataset 44
5.4 Similar Images Produce Different Results by Above Water Detector, Left: Correctly Detected, Right: Similar Image Displaying All Predictions . . . 45
5.5 Similar Images Produce Different Results by Below Water Detector, Left: Correctly Detected, Right: Similar Image Displaying All Predictions . . . 46
5.6 Similar Images Produce Different Results by Hybrid Detector, Left: Correctly Detected, Right: Similar Image Displaying All Predictions 47
5.7 IOU Might be Too Strict, Left: Hybrid, Middle: Below Water, Right: Ground Truth 48
5.8 IOU Might be Too Strict, Left: Hybrid, Middle-Left: Below Water, Middle-Right: Above Water, Right: Ground Truth 49
5.9 Object Only Found by Below Water Dataset, Left: Hybrid, Right: Below Water 50
5.10 Object Only Found by Below Water Dataset, Left: Hybrid Dataset, Right: Below Water Dataset 51
5.11 Example Images from the Validation Dataset 52

List of Tables

5.1 Manual Labelling Time for Above Water Dataset 40
5.2 Manual Labelling Time for Below Water Dataset 40
5.3 Manual Labelling Time for Hybrid Dataset 41

Abbreviations

AI Artificial Intelligence

AP Average Precision

AUV Autonomous Underwater Vehicle

CNN Convolutional Neural Network

COCO Common Objects in Context

CPU Central Processing Unit

DL Deep Learning

DOF Degrees of Freedom

FLOPS Floating Point Operations

FN False Negative

FP False Positive

FPS Frames Per Second

GPU Graphics Processing Unit

IMR Inspection Maintenance and Repair

IOU Intersection Over Union

MIT Massachusetts Institute of Technology

ML Machine Learning

NTNU Norwegian University of Science and Technology

R-CNN Regional Convolutional Neural network

ROV Remotely Operated Vehicle

TP True Positive

VOC Visual Objects in Context

YOLO You Only Look Once

Chapter 1

Introduction

1.1 Background and Motivation

In a continuously developing world of automation and data driven science, the question arises of how to implement these new functions in a marine environment. Eelume is a company that has envisioned a future where Autonomous Underwater Vehicles (AUVs) are stationed throughout the ocean floor[1] as seen in figure 1.1. The AUVs are equipped with manipulators and sensors to perform Inspection, Maintenance and Repair (IMR) on subsea installations.

FIGURE 1.1: Autonomous Underwater Vehicles Performing Inspection[1]

This contrasts to the standard method of performing IMR today, which involves a Remotely Operated Vehicle (ROV) as shown in figure 1.2. These operations are large in nature and typically cost 100-300 thousand USD per day[2]. One of the main reasons for this high cost is because the ROV is connected to a vessel on the surface. The connecting cable, known as the umbilical, is responsible for power, video stream and communications. Although it only requires two people to operate an ROV, the ship crew typically consists of 20-30 people who take part in the high operational expenses.

The aforementioned AUV solution opens industry minds and welcome a sought after cost reduction.

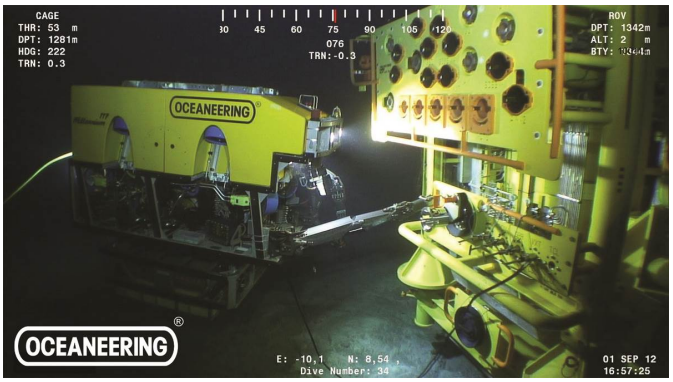


FIGURE 1.2: Remotely Operated Vehicle Working on a Subsea Installation[3]

1.2 Object Detection

Some examples of what an AUV is expected to do includes:

- Inspecting long stretches of pipes
- Moving valves by rotation or hot stab
- Inspecting general equipment on the seabed
- Recording information on video

If an underwater vehicle are to do these things autonomously it is inherent that it knows what it is looking at. The on-board cameras and sensors will be the basis for decision making and might extend to control of the vehicle as well[4][5]. Deciding which valve to turn at any given time is critical to the overall operation and has tremendous repercussions if not done correctly. With these factors in mind it is therefore decided to look closer into how one can know or categorize what is currently being perceived by the on-board camera of an AUV. Such an endeavour is known as object detection and defined in this report by Amit and Pedro[6] as follows:

> *Object detection involves detecting instances of objects from a particular class in an image.*

1.3 Earlier Work

The ideas of machine vision and object detection have been around for a while. Many regard the work done by LeCun et al.[7] from 1989 as the first breakthrough for deep learning and smart algorithms. They managed to recognize handwritten zip codes from the United States by using a computer. The field has however, evolved at a slow pace until fairly recently, what changed? Essentially there were two factors: bigger datasets and higher computational power. This allowed deep learning to enter the stage of image classification and object detection, and its success is one of the main motivations for using deep learning in this report. Presented below are some key developments on the path to state-of-the-art performance within the field. Note particularly the deep learning revolution that started in 2012, as described in section 1.3.3.

1.3.1 Viola-Jones Algorithm

The first key development came from Paul Viola and Michael Jones in 2001 and is aptly named the Viola-Jones Algorithm[8]. It included a demo where it was possible to detect faces in real time on a webcam feed. They took advantage of the fact that when converting an image into grayscale, the eye region is typically darker than the area around cheeks and nose. By using clever mathematics this was hardcoded into a formula and calculated on a Central Processing Unit (CPU). A demonstration of the feature extraction is shown in figure 1.3, where the top row displays features, while the bottom row shows an overlay on the actual image. The first feature defines where the eyes are, whereas the second feature finds the nose and then scaling is done accordingly to detect an entire face. Although this was stunning at the time, it was limited to how these features were set up by the authors, and could for example not detect a rotated face.

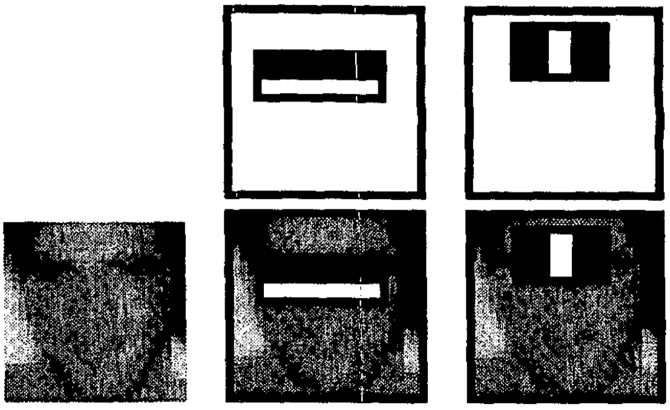


FIGURE 1.3: Main Features of the Viola-Jones Algorithm[8]

1.3.2 Histogram of Oriented Gradients

The field was further developed when in 2005, Dalal and Triggs[9] published their method: Histograms of Oriented Gradients, or HOG. It was more effective than current methods in the sense that it was faster and had a lower error rate. The method is centered around gradients and essentially measures how dark the current pixel is compared to the surrounding pixels. The gradient is then calculated to be a vector pointing in the direction of where the image turns darkest. Each pixel is replaced by a gradient and gathered in a small square with 16x16 pixels, where the vector with the largest magnitude is chosen. This results in a simple representation that capture the basic structures of a face, as can be seen from figure 1.4.

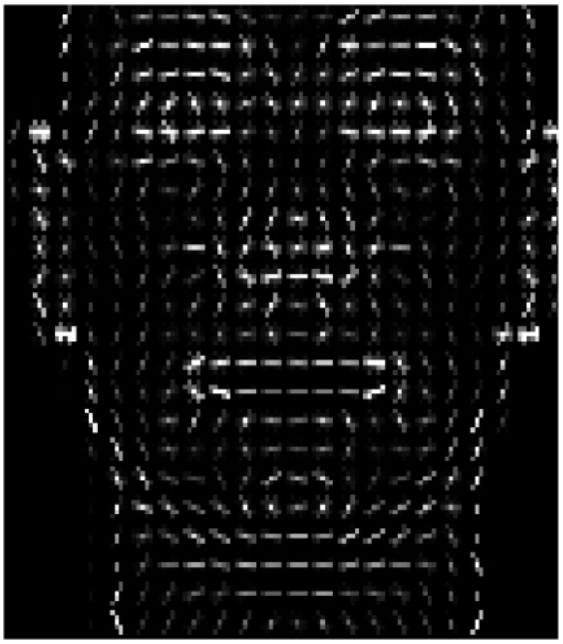


FIGURE 1.4: Representation of a Face Using Histogram of Oriented Gradients[9]

1.3.3 The Deep Learning Revolution

In 2012 the annual ImageNet[10] competition, where software developers compete to correctly classify objects, was stunned to see that a successful implementation of a deep neural network had outperformed all other candidates, including the previous year, by 40% in accuracy. This is widely considered as the beginning of the deep learning revolution and put neural networks on the radar for a big part of the tech industry. The winning contribution was made by Krizhevsky et al.[11] and used a convolutional neural network to classify images with a single label. Figure 1.5 shows eight test images with their labels and the five most probable categories predicted by their algorithm.

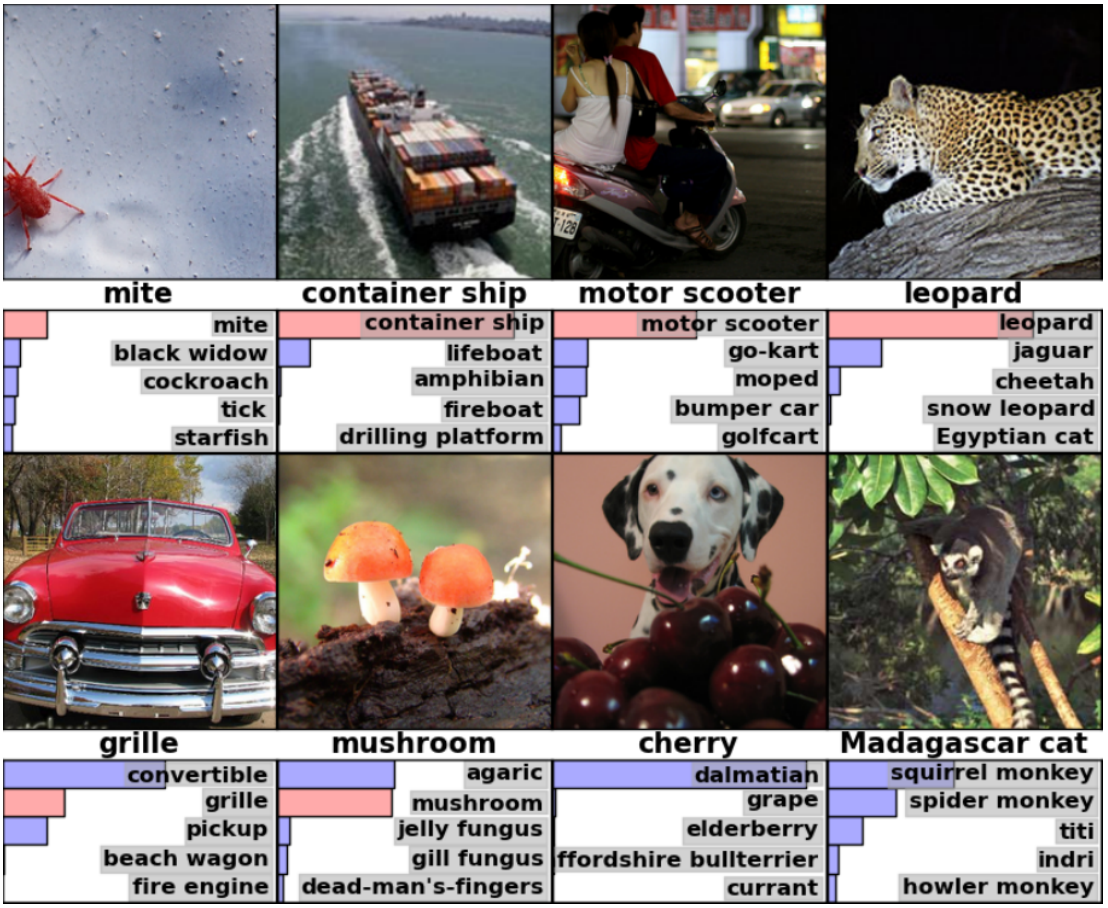


FIGURE 1.5: Test Images with Labels and Probability Distribution From Krizhevsky et al.[11]

1.3.4 R-CNN

In the wake of the deep learning revolution, it was no longer sufficient to simply classify an object, but specifically there was a commitment to raise the bar by detecting several objects *and* their relative positions in an image, also known as object detection. This was stimulated by the yearly Pascal Visual Object Classes (VOC) Challenge[12] created in 2010, which is a dataset that offers more than 11 thousand annotated images with 27 thousand individual objects divided over 20 classes. Over the years, this challenge and its associated dataset has become accepted as *the* benchmark for object detection. It was a challenging competition which proved to be quite difficult and did not have much progression until Girshick et al. presented Regional - Convolutional Neural Network (R-CNN)[13] in 2013, which outperformed all current detection algorithms by 30% with a mean Average Precision (mAP) of 58.5%.

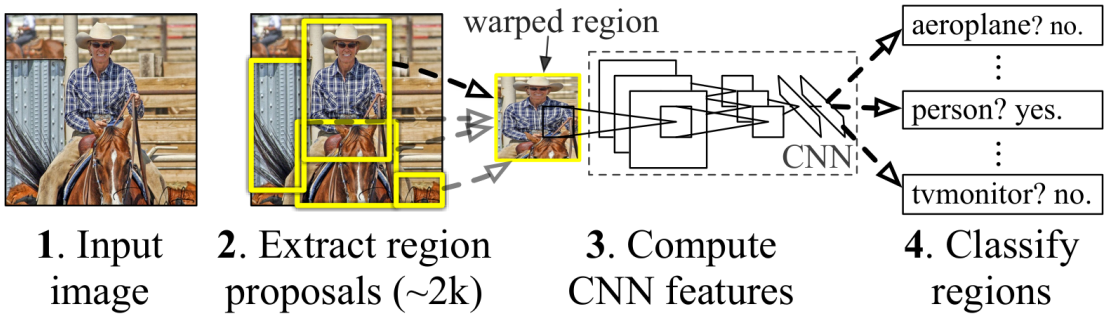


FIGURE 1.6: System Overview of R-CNN Algorithm[13]

Their approach is shown in figure 1.6 and is based on four steps. 1) Take an input image, 2) extract around 2000 region proposals, 3) send these region proposals of the image to a convolutional neural network, 4) classify each region by using a standard classifier. It proved to be an effective combination based on the aforementioned results, but the major drawback was that it took around 13 seconds per image on a powerful Graphics Processing Unit (GPU). This was shortly addressed when the upgraded versions: Fast R-CNN[14] (2015), Faster R-CNN[15] (2015), and Mask R-CNN[16] (2017) were introduced. The final result was a detector which ran steadily at 7 frames per second (FPS) with mAP of 73.2% on a powerful GPU.

1.3.5 YOLO

The YOLO (You Only Look Once) algorithm[17] takes a different approach than previous detectors. By exclusively using a Convolutional Neural Network (CNN), it bypasses any region proposals or standard classifiers, meaning it detects and classifies objects in one shot (hence; You Only Look Once). When originally proposed in 2015 it boasted not of its accuracy in the Pascal VOC Challenge, of 64.4% mAP, but the low runtime which allowed 45 FPS. This was subsequently improved in 2016 when the authors revised the model to YOLOv2[18], which yielded 78.6% mAP at 40 FPS. A third paper was published in 2018, called YOLOv3[19], but offered little improvement in accuracy. The main focus was on several features that were tested, but did not work. Nonetheless, YOLOv2 and YOLOv3 stands as the current state-of-the-art in real-time object detection because of their accuracy and speed. Figure 1.7 displays a sample of diverse objects being detected by YOLOv2.

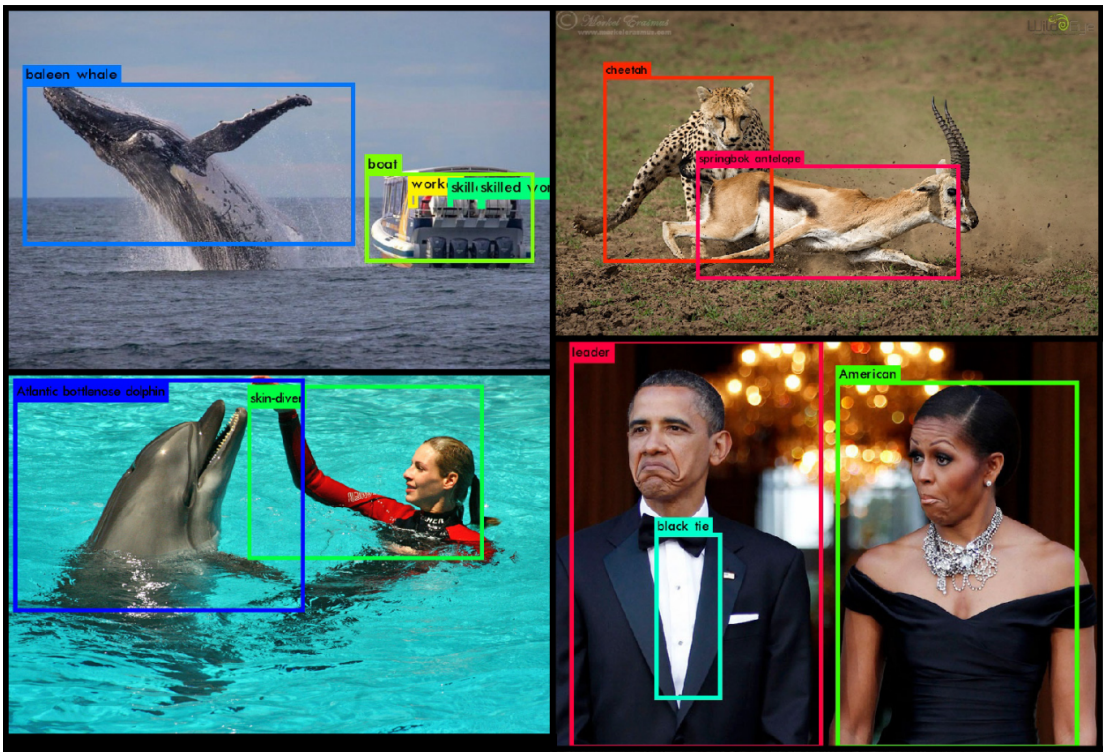


FIGURE 1.7: Object Detection Samples from YOLOv2[18]

1.3.6 Underwater Visual Tracking

Although fairly recent, the YOLO algorithm has gained widespread success and is being used from applications in defense industry[19] to Zebra censusing[20]. For this report, the most interesting implementation was visual tracking of an underwater robot using object detection by Shkurti et al.[21], from September 2017. They propose a scaled down version of the YOLOv2 architecture, that is capable of running on the limited hardware, to track an underwater robot in an unstructured 3D environment. The object detector was trained on a dataset of hand-annotated images of an Aqua hexapod robot and later used in a real-world scenario to follow another robot's independent motion in 5 Degrees of Freedom (DOF). In total they used 5200 images to train their network.

FIGURE 1.8: Hexapod Underwater Visual Target Tracking Using YOLOv2[21]

1.4 Thesis Objective

This thesis seeks to investigate the use of deep learning as an apparatus for robust object detection through images in underwater robotics. Traditional computer vision is error-prone due to manual feature extraction, while deep learning, specifically Convolutional Neural Networks, provide greater stability against unforeseen disturbances. A real-time object detector that can be custom trained on any image dataset captured through a simple ROV camera is presented. Such an object detector can be customized to detect any object, and applied to navigation and decision support in subsea operations.

A hammer was chosen as the detectable object because of possible tool selection by a subsea AUV, but it is used to showcase the flexibility of the method. Images are gathered by a BlueROV2 drone and the thesis examines differences between three datasets, here divided into: below water, above water and hybrid, which is a combination of the two. The interest in a hybrid dataset arises from the fact that above water images are easier, cheaper and more plentiful to come by in most circumstances. While on the other hand, below water images are from the operational domain and necessary for a functional object detector. The report goes through how one creates these datasets, implements a state-of-the-art convolutional neural network and lastly how these datasets compare on the same benchmark. In figure 1.9, this is represented on the left side as the *Training Phase* of an object detector. While the *Deployment Phase* describes how an object detector would be implemented in real-time on an ROV, and is not part of the scope for this report.

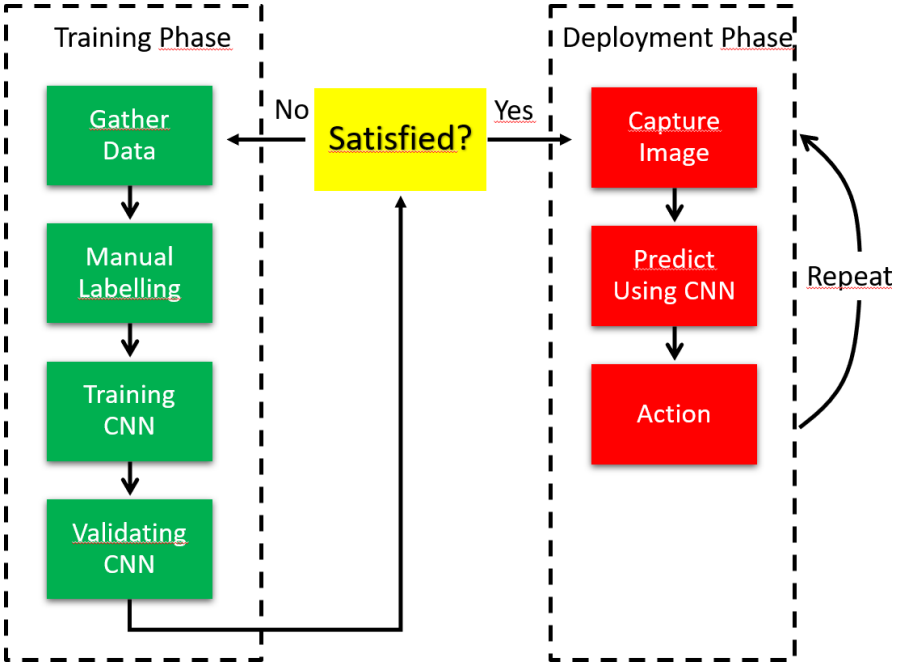


FIGURE 1.9: CNN Object Detector Flow Chart

1.5 Thesis Structure

Chapter 2 gives an introduction to some of the relevant theory to better understand how a deep learning network works, with specific emphasis on CNNs. In addition it goes into detail on how the YOLO framework is set up and which evaluation metrics that are used to compare different training regimes.

Chapter 3 explains how the Marine Cybernetics lab at NTNU was set up, together with how the image datasets were collected, as well as manually labelled.

Chapter 4 describes the method used to implement YOLO on a personal computer and goes through some useful tools and code that make it practical to train and run.

Chapter 5 presents the results from manually labelling all the images, results from validating the three object detectors and a few intriguing detection examples.

Chapter 6 discusses the results.

Chapter 7 draws a conclusion based on the results and discussion, in addition to proposing further work.

Chapter 2

Theory

2.1 Deep Learning

Terms like artificial intelligence (AI), machine learning (ML) and deep learning (DL) are liberally thrown around in various settings. It is therefore of interest to take a deeper dive in order to understand the capabilities and limitations of these technologies. The definitions of AI, ML, and DL are gathered in a hierarchical nature in figure 2.1, and proposed below.

Artificial Intelligence:

> *"Any technique which enables computers to mimic human behaviour."*

Machine Learning:

> *"Subset of AI that employs statistical methods for machines to improve with experience."*

Deep Learning:

> *"Subset of machine learning that utilizes multi-layer perceptrons."*

2.1.1 Supervised Learning

Supervised learning is one way to train deep learning algorithms and is fundamentally a function approximation problem. Given a dataset, it is of interest to find a function

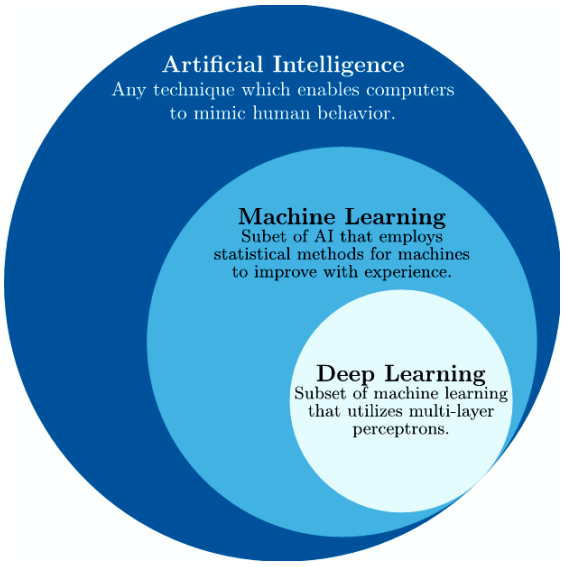


FIGURE 2.1: Hierarchical Structure of Artificial Intelligence, Machine Learning and Deep Learning[22]

that can relate input to output. This is done by computing an error between desired and predicted output, and applying this information to tune the unknown function. An overview of how this looks like can be seen in figure 2.2, where the target y and input x is known, and the goal is to minimize the difference between predicted $\hat{y}$ and target y. This requires large amounts of data and can be extremely CPU or GPU intensive.

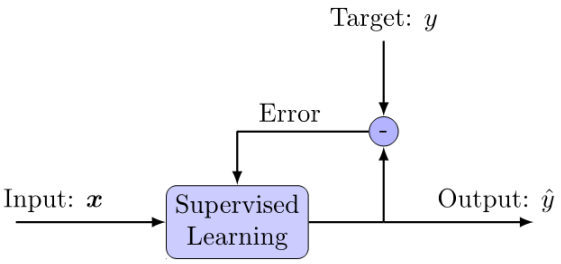


FIGURE 2.2: Supervised Learning Overview[22]

2.2 Deep Feedforward Networks

Deep feedforward networks are the cornerstone when it comes to deep learning. Also known as feedforward neural networks, they are loosely inspired by the brain. More specifically, there are nodes that send information forward, based on their input. Figure 2.3 displays a network with input on the left and output on the right. In between there are so-called *hidden* layers, which may consist of one (shallow) or multiple (deep) layers. The arrow from one node to another represents a weight, meaning a number which is multiplied by the value coming out of each node. These weights are the main tuning parameters and is where learning manifests itself during training. Note that in the figure,

each node in the hidden layer is connected to each input and each output node. This is what makes the network complex, but also what makes it so powerful. It represents the foundation of any neural network is be the benchmark on how to differentiate other network structures in this report.

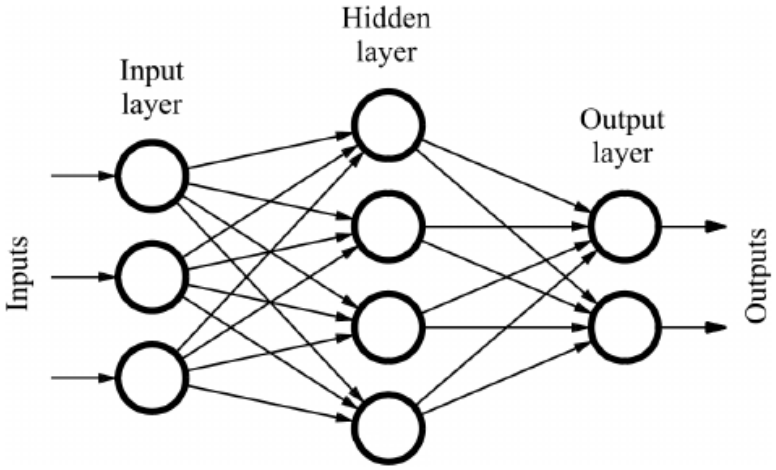


FIGURE 2.3: A Fully Connected Feed Forward Network[23]

By looking closer into the nature of a network, one realizes that in essence, each layer consists of matrix multiplication[24]. Consider the value of each node collected in a vector $\boldsymbol{x}$, which subsequently gets multiplied by a matrix $\boldsymbol{W}$, that contains all the weights between each node. In addition there may be a bias vector $\boldsymbol{c}$, which can be considered as the zeroth weight. This yields the function $f(\boldsymbol{x}, \boldsymbol{W}, \boldsymbol{c})$.

$$f(\boldsymbol{x}, \boldsymbol{W}, \boldsymbol{c}) = \boldsymbol{W}^T \boldsymbol{x} + \boldsymbol{c} \tag{2.1}$$

This linear combination is easily chained together, making it possible to create a deep network with many layers, hence $f(\boldsymbol{x}) = \boldsymbol{f_1}(\boldsymbol{f_2}(\boldsymbol{f_3}(\boldsymbol{x})))$. However, since a combination of linear matrix multiplications is itself linear, there is something missing if the interest is to solve nonlinear problems. This is where an activation function is important.

2.3 Activation Function

The power of deep learning algorithms to solve nonlinear problems is based on the activation function, which is here denoted as $g(x)$. It can be proven that if the activation function is nonlinear, neural networks can represent any continuous function on a subset of $\mathbb{R}^n$. Although not feasible to train such a network, it was first proven by Cybenko[25] using a single hidden layer and a sigmoid activation function. This was later expanded by Hornik[26] to include any bounded and non-constant activation function. Thus, they

are crucial to the deep learning algorithm. A series of plausible functions exist, but they can be formally described in the following manner, where $\boldsymbol{h}$ is used to represent the vector output of a hidden layer.

$$\boldsymbol{h} = g(f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c})) = g(\boldsymbol{W}^T \boldsymbol{x} + \boldsymbol{c}) \tag{2.2}$$

2.3.1 Rectified Linear Unit

The rectified linear unit, ReLU, is essentially a linear function, except it removes all negative values. Note that its derivative is not defined for $x = 0$.

$$g(x) = max(0, x) \tag{2.3}$$

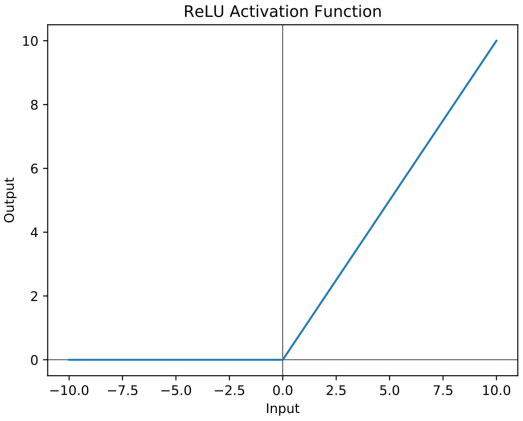


FIGURE 2.4: Rectified Linear Unit Activation Function[24]

2.3.2 Leaky Rectified Linear Unit

The leaky rectified linear unit, leaky ReLU, is very similar to the ReLU function, except that it has a nonzero derivative for negative inputs.

$$g(x) = \begin{cases} x, & if\ x > 0 \\ 0.1x, & otherwise \end{cases} \tag{2.4}$$

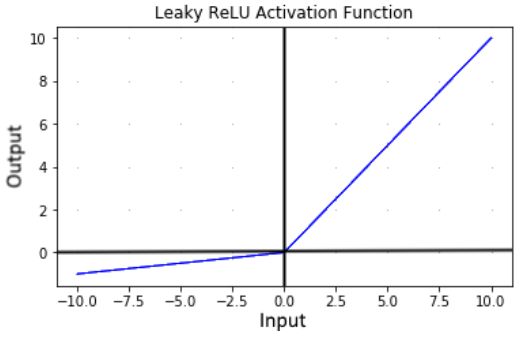


FIGURE 2.5: Leaky Rectified Linear Unit Activation Function[24]

2.3.3 Sigmoid Activation Function

The sigmoid function maps input to the range (0, 1), is continuously differentiable and monotonically increasing.

$$g(x) = \frac{1}{1 + e^{-x}} \qquad (2.5)$$

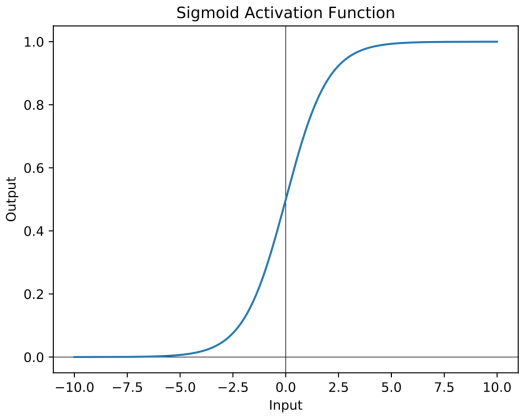


FIGURE 2.6: Sigmoid Activation Function[24]

2.3.4 Tanh Activation Function

The hyperbolic tangent function maps input to the range (-1, 1), is continuously differentiable and monotonically increasing.

$$g(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \qquad (2.6)$$

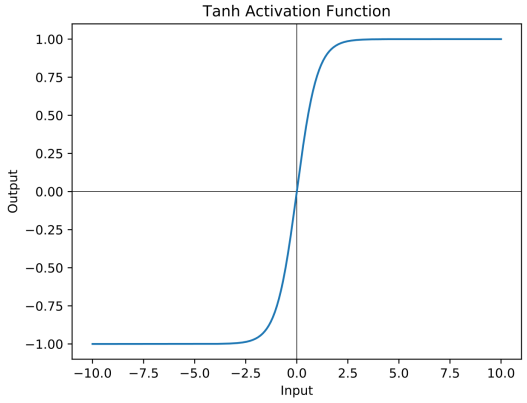


FIGURE 2.7: Tanh Activation Function[24]

The activation functions mentioned above are only a subset of the available functions and they come in several variants. For example, a modified version of the rectified linear unit called Exponential Linear Unit (ELU), which is differentiable on its entire domain. The ReLU however, has become increasingly popular lately, mainly because of its fast computational speed.

2.4 Output Function

After a specified number of hidden layers and activation functions, the final step consists of the output layer. For classifiers it is common to use a softmax function which normalizes a vector into a probability distribution. It takes a vector with arbitrary numbers

and returns a value in the range of 0 to 1 for each element, in addition the sum of all elements have to equal 1. For a vector $\boldsymbol{z}$ over K classes, the jth element is defined as follows.

$$\sigma(\boldsymbol{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \tag{2.7}$$

This is very useful when dealing with several classes where the goal is to identify the most likely probability. For example if you know that there are 20 classes, but there is only one that can be correct. If on the other hand, you want to do a binary classification problem where the answer would be of the type yes/no or pass/fail, the sigmoid function is recommended. As defined for an element in section 2.3.3, it also yields an output between 0 and 1, where 0.5 is the divider between outcome one or two.

2.5 Optimizing With Gradient Descent

When the output is calculated through all the layers, it is then compared with the ground truth and an error is calculated. More specifically a loss function, $\boldsymbol{L}$, is defined for the given problem. Where the squared Euclidean norm, also known as the squared spatial distance, between predicted and actual ground truth, is a typical choice. During training, the error is used to determine how each weight in the network should be changed. Visually described for a single weight in figure 2.8, it can be thought of as a ball (weight) that is seeking the lowest value of a path (cost function). At each iteration the weight is moved in the direction that reduces its cost function the most, found from the gradient, and moves a predefined length known as the learning rate. Such a gradient is found through a method known as back propagation which applies the chain rule[27].

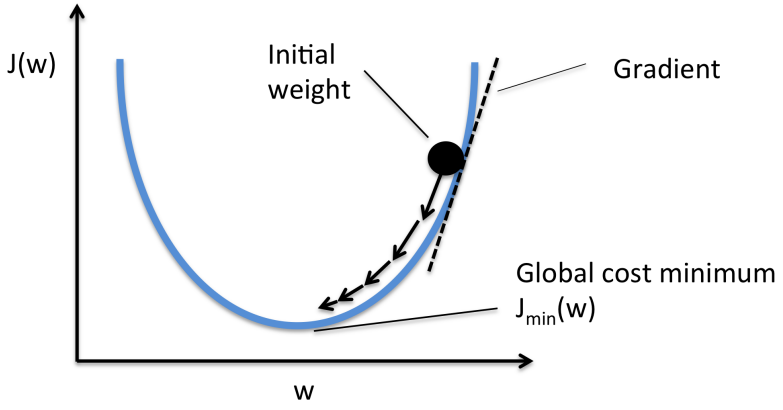


FIGURE 2.8: Gradient Descent Illustrated as a Ball Seeking Lowest Position of a Path[28]

2.6 Convolutional Neural Networks

Convolutional Neural Networks are a specialized type of neural networks which uses a method called convolution in place of general matrix multiplication. Thus the hidden layers are called convolutional layers instead. Figure 2.9 shows the typical stages in a CNN layer. After the convolution is completed, a nonlinear activation function is applied and pooling is performed, see section 2.6.5. The architecture is based on the previously mentioned feedforward network shown in figure 2.3, but incorporates a few different ideas which make CNNs unique and effective. Amongst these are *sparse interactions*, *parameter sharing* and *equivariant representations*. These let the network exploit local structures and make it significantly more effective than a traditional network for large dimensional inputs.

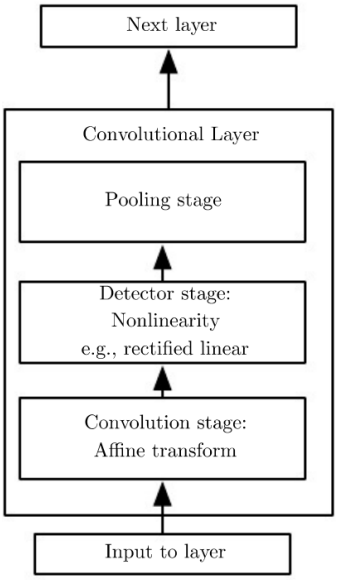


FIGURE 2.9: Typical Stages of a Convolutional Neural Network Layer[24]

2.6.1 Convolution

Convolution is a mathematical operation of two functions, f and g, denoted $f * g$. It is a particular form of integral transform and is defined as follows:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau \tag{2.8}$$

However, this formula is not very intuitive and therefore a visual understanding is sought for. Another way of looking at convolution with respect to deep learning is given by figure 2.10, where the convolution, denoted $\mathbf{I} \circledast \mathbf{K}$, is rather thought of as the dot product

between an image I, and a kernel K. The kernel, also known as a filter, slides over the image and produces a single value for each input.

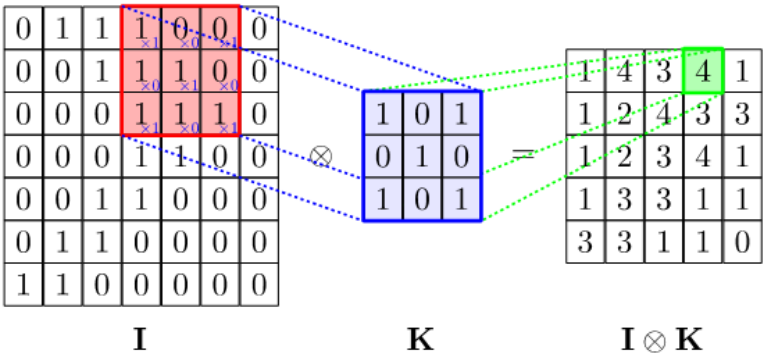


FIGURE 2.10: Convolution as the Dot Product Between Kernel and Input[22]

These kernels contain unknown weights and are tuned based on the desired result. They can be initialized with random numbers, or take a specific form which may be known to produce a desired outcome. An example can be seen in figure 2.11, where an edge detector kernel has been used on an image. The result is known as a feature map, and shows where the kernel finds an edge (light) and where it does not (dark).

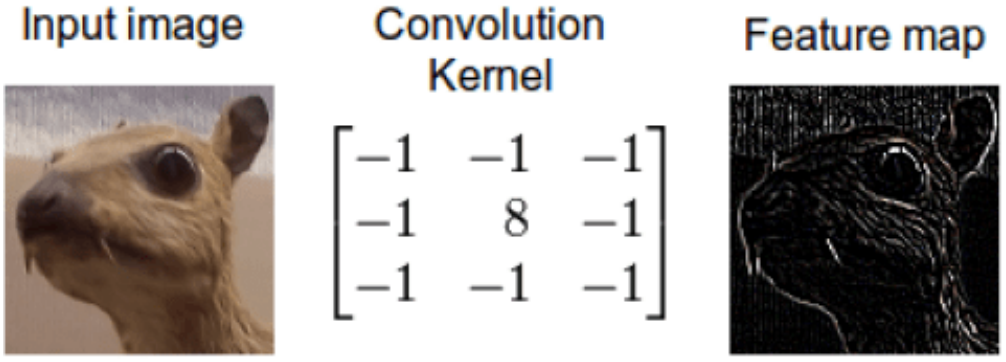


FIGURE 2.11: Convolution Using an Edge Detection Kernel[29]

2.6.2 Sparse Interactions

The convolution operation only interacts with a portion of the image at each time, taking advantage of an idea called sparse interactions. Compared to a fully connected network, where the kernel would have the same size as the input, this yields lower storage and computational cost. Figure 2.12 shows a representation of how sparse interactions can look like for an arbitrary layer. Using x as inputs and s as outputs with interactions of the middle node highlighted in grey.

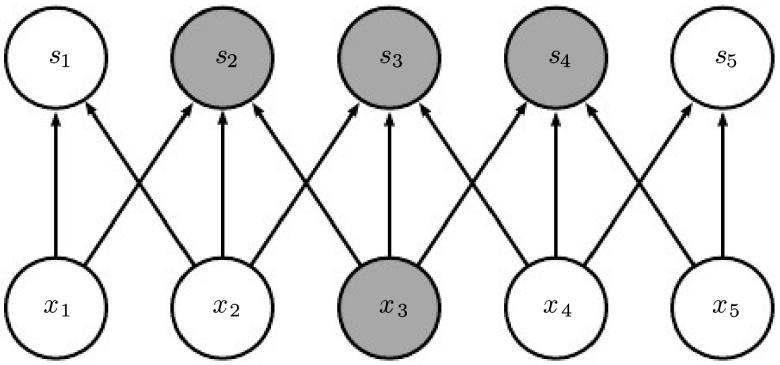


FIGURE 2.12: Sparse Interactions Between Nodes[24]

2.6.3 Parameter Sharing

Since the same kernel is used for the entire input, there is no need to learn new parameters for each operation. The size and amount of kernels, are universal and to be tuned. A kernel is in a sense "looking" for something specific, for example an edge. This is called parameter sharing and does not reduce actual runtime when comparing to a traditional feedforward network, but reduces the number of weights to be stored. Thus saving on training time and memory. An example of parameter sharing is shown for an arbitrary layer in figure 2.13, where an output node which uses two neighboring nodes as inputs, have the same weights, a and b, for each operation. In a traditional feedforward network, all the weights would be unique and independent, requiring vastly more storage.

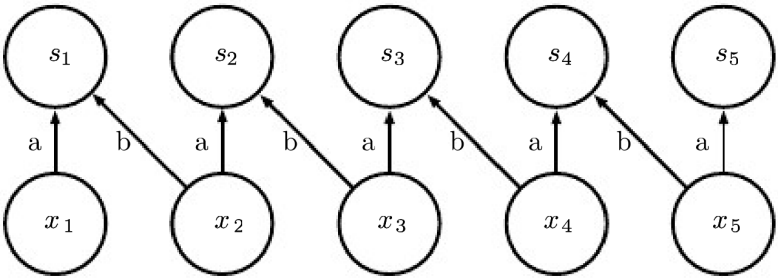


FIGURE 2.13: Parameter Sharing Between Nodes[24]

2.6.4 Equivariant Representations

A powerful property of CNNs is that they are equivariant. This arises from the convolution operation and means that if the input changes, the output changes accordingly. The way convolution is set up, it causes the layers to have equivariance to translation automatically. Thus if an edge moves sideways it will have the same representation in the output, but changed a similar distance as the input. However, it is not equivariant to other transformations, like rotation, by itself.

2.6.5 Pooling and Downsampling

After the nonlinear activation function is applied to convolutional outputs, an operation known as pooling modifies the values even further. A pooling operation is typically an average, or the maximum value on a set number of nodes. Since pooling summarizes several neighboring nodes, it is natural to use fewer pooling units. This causes a reduction in the number of nodes and is also known as downsampling. Figure 2.14 shows how max pool and downsampling is used together on a layer, where the pooling width is three and stride between pools is two.

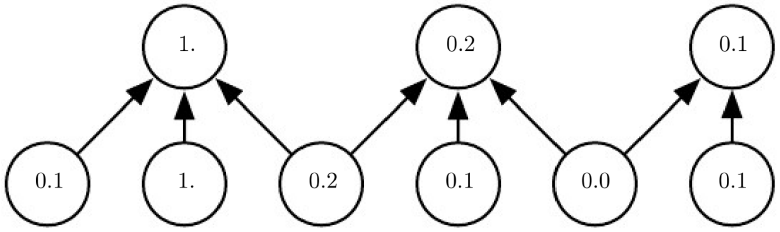


FIGURE 2.14: Max Pool and Downsampling Between Nodes[24]

Besides the reduction in memory requirements, one motivation for applying these features is invariance. A good example is how pooling causes invariance to rotation, as can be seen in figure 2.15. As mentioned previously, the convolutional operation is naturally invariant to translation, and pooling is added to enhance this feature in other domains as well.

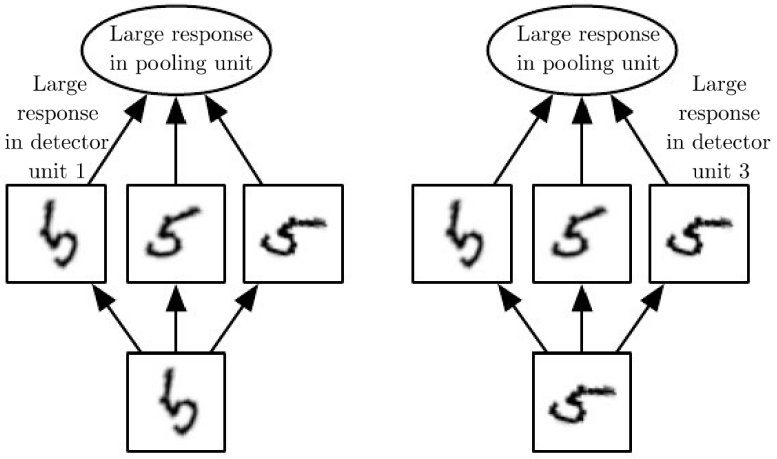


FIGURE 2.15: How Pooling Causes Rotation Invariance[24]

2.7 Evaluation Metrics

In order to evaluate performance concerning trained models, it is of importance to define metrics that can compare different object detection methods. Although there are

varying parameters for different competitions, the Average Precision (AP) as defined in the Pascal VOC Challenge[12], will be used in this report. AP is calculated by integrating the Precision x Recall curve, while the precision and recall are calculated from the correct detections defined by a threshold value of Intersection Over Union (IOU). All of these terms are further explained below.

2.7.1 Intersection Over Union

The Intersection Over Union (IOU) parameter is a basic building block in the toolkit for object detection evaluations. It calculates the overlap with the ground truth bounding box B_{gt}, and a predicted bounding box B_p, and divides it by the area of union between them. This is illustrated in figure 2.16, where the ground truth bounding box is green and the detected bounding box is red, it is also defined in equation 2.9 below. Following competition rules, the IOU has to exceed 0.5 in order to be considered a correct detection.

$$IOU = \frac{area(B_p \cap B_{gt})}{area(B_p \cup B_{gt})} \tag{2.9}$$

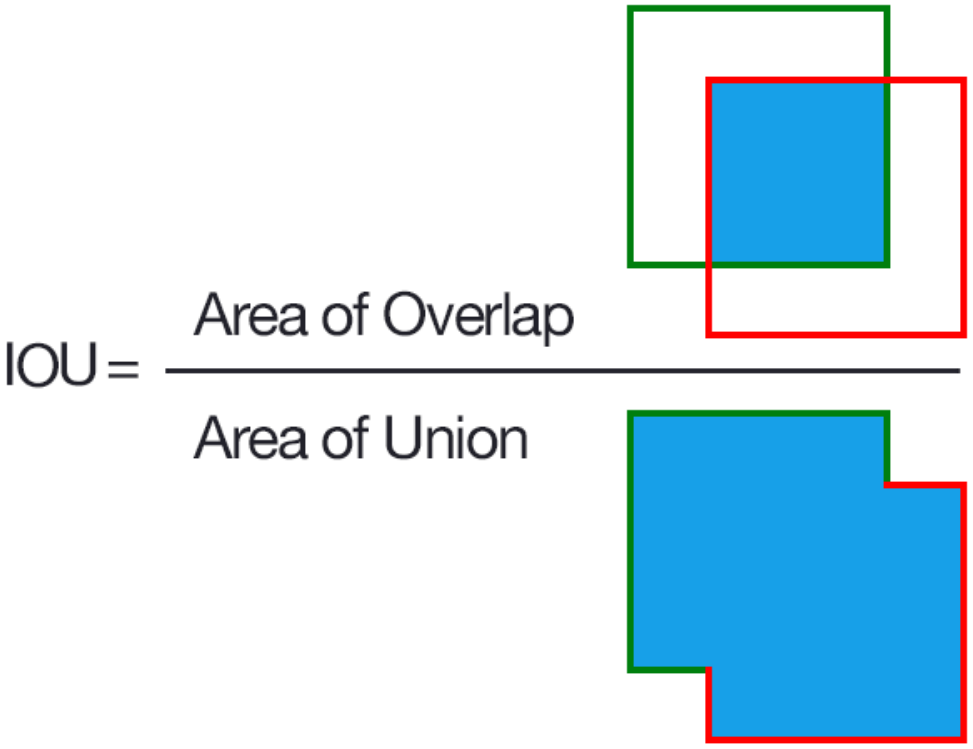


FIGURE 2.16: Intersection Over Union[30]

2.7.2 Precision x Recall Curve

Before delving into the definitions of precision and recall, it is worth to describe some basic concepts.

- **True Positive (TP)**: A correct detection. Detection with IOU ≥ 0.5
- **False Positive (FP)**: A wrong detection. Detection with IOU < 0.5
- **False Negative (FN)**: A ground truth not detected.

Precision is the ratio of how many correct predictions the model presented. A value of 1.0 means that all predictions were correct. It says nothing about how many predictions were provided overall and it can therefore be the case that there were very few predictions in total. Precision is given by:

$$Precision = \frac{TP}{TP + FP} = \frac{TP}{all\ detections} \tag{2.10}$$

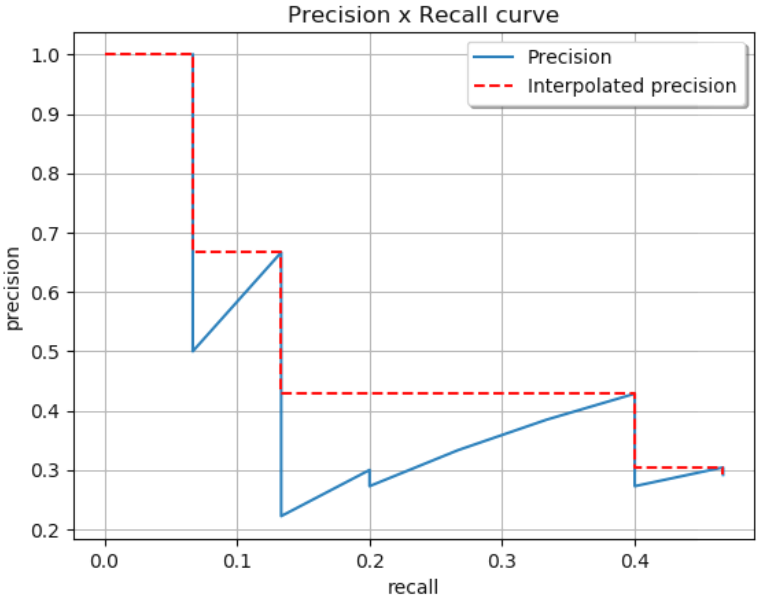


FIGURE 2.17: Precision x Recall Curve[30]

Recall is the ratio of how many objects were correctly detected by the model. A value of 1.0 means that all objects in the validation dataset have been identified. It says nothing about how many wrong predictions were made and it can therefore be the case that there were several wrong predictions also. Recall is defined as:

$$Recall = \frac{TP}{TP + FN} = \frac{TP}{all\ ground\ truths} \tag{2.11}$$

An excellent object detector would have high precision *and* high recall. The metrics are calculated for each prediction in the validation dataset and plotted to yield the Precision x Recall curve. An example of how such a curve looks like is shown in figure 2.17 for a dataset of 7 images with 15 predictions.

2.7.3 Average Precision

The main reason for explaining these parameters mentioned above is because of Average Precision, which might be the most important metric. It is found by integrating the Precision x Recall curve for interpolated values. This is illustrated in figure 2.18, such that AP = A1 + A2 + A3 + A4. Average precision ranges from 0-100% and is set up in such a way that 100% means that all objects have been detected correctly and that no false predictions were made. The metric is easily transferable to different classes of objects which is useful when there can be upwards of 500 classes. If one deals with more than one class the standard is to average over all the calculated APs, known as mean Average Precision (mAP).

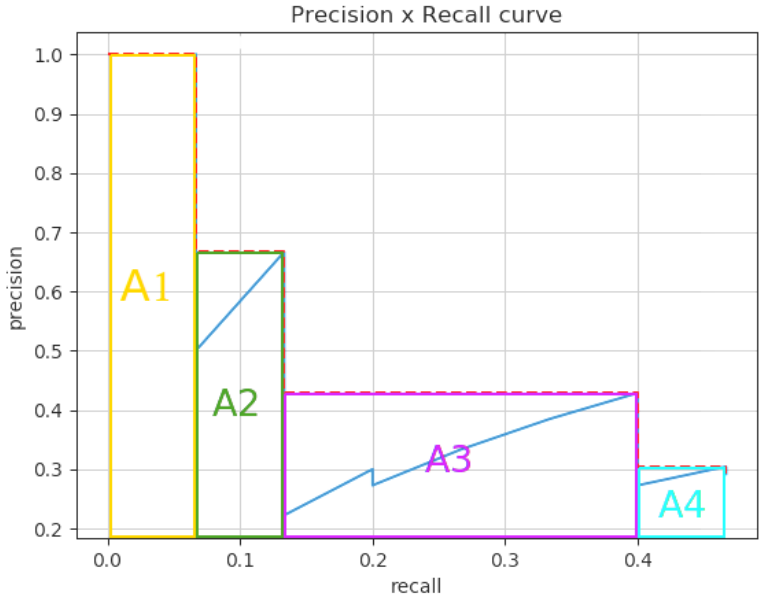


FIGURE 2.18: Average Precision from Integrating the Precision x Recall Curve[30]

2.8 YOLO

As mentioned previously, the second[18] and third[19] iteration of YOLO are currently state-of-the-art in object detection. Despite the fact that YOLOv3 is newer than YOLOv2, it adds little of value, something which is also alluded to in the title: *"Yolov3:*

An Incremental Improvement." In addition, it is worth noting that the YOLOv2 framework is considerably smaller allowing it to run faster than YOLOv3, and is therefore preferable for real-time detection on limited hardware. There are also examples that demonstrate the usage of YOLOv2 underwater, see section 1.3.6. Based on the Project Thesis of the author[31] and the differences explained in this section, together with the fact that there is substantial documentation online, leads this thesis to continue with YOLOv2.

The algorithm offers a CNN architecture called Darknet-19 which can be seen in figure 2.19. There is a total of 19 convolutional layers (hence Darknet-19), where each contains the leaky rectified linear unit (leaky ReLU) activation function, as defined in section 2.3.2, and maxpool with size 2x2 and stride 2. The final stage consists of an average pool before a fully connected layer and a softmax to find the probability distribution over all classes.

Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2/2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2/2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2/2$	28×28
Convolutional	256	3×3	28×28
Convolutional	128	1×1	28×28
Convolutional	256	3×3	28×28
Maxpool		$2 \times 2/2$	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Maxpool		$2 \times 2/2$	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	1000	1×1	7×7
Avgpool		Global	1000
Softmax			

FIGURE 2.19: Architecture of Darknet-19[18]

2.8.1 Prediction Tensor

To better understand YOLO it is of interest to see how the output is configured. As can be seen in figure 2.20, the image is divided into SxS grid cells. Each cell is responsible for predicting any object that may lie inside it. It does this by predicting B bounding boxes, as well as C class probabilities. A bounding box has five components: x, y, w, h

and *confidence*. The x, y coordinates are for the center of the box , while w, h are width and height, respectively. Even though there is only one grid cell responsible for predicting any object that falls inside it, the bounding boxes are fully capable of extending beyond the size of a cell. The confidence score is defined as the IOU between a predicted box and the ground truth. It is indifferent to which class the object belongs to, and gives a score of zero if there are no objects present. Since there are five bounding box components and each cell makes B of those, the output tensor is now of shape SxS with a depth of B*5.

In addition, there are class probabilities to predict. These are conditioned on the grid cell containing an object, meaning that they are predicted for each cell independent of bounding boxes. It also means that the loss function does not penalize the detector for predicting a wrong class when there is no object present in the cell, as explained below. Only one set of class probabilities are predicted per cell, thus the final shape of the output tensor is SxS with depth B*5 + C. For this report, $S = 7$, $B = 2$ and $C = 1$ so that the output tensor is 7x7 with a depth of 11.

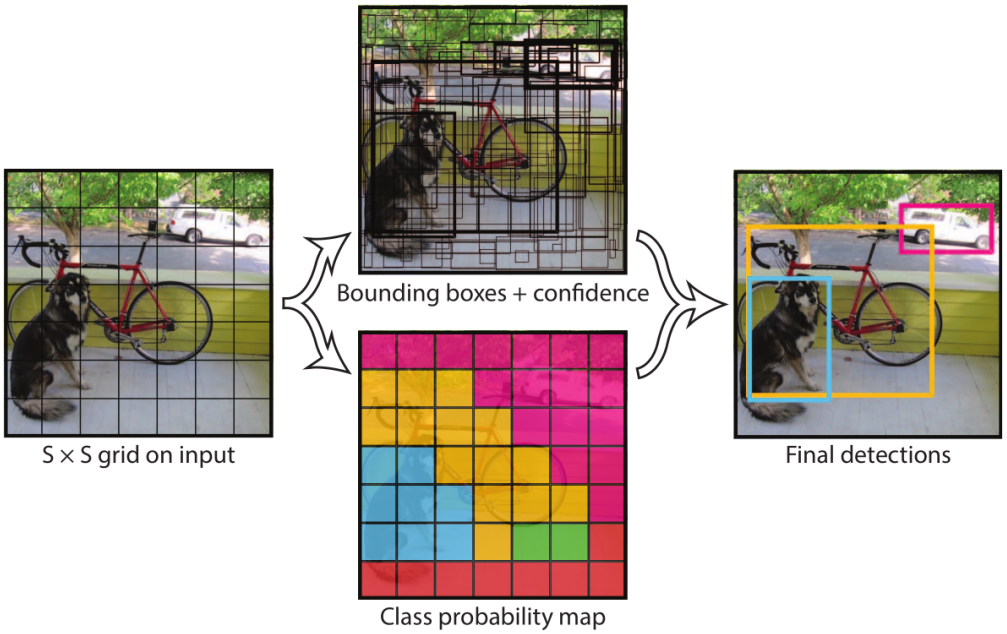


FIGURE 2.20: System Overview of YOLO Algorithm[17]

2.8.2 Loss Function

The YOLO loss function determines what the neural network uses as a basis for learning. Also known as an objective function, it is what the network will try to minimize through gradient descent, see section 2.5. The loss function for YOLO is quite complex, but will be further explained in this section, the complete function can be seen in equation 2.12.

$$
\begin{aligned}
&\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&+\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} (C_i - \hat{C}_i)^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{noobj} (C_i - \hat{C}_i)^2 \\
&+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{noobj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2
\end{aligned}
\tag{2.12}
$$

Before delving into the individual terms, there are a few common features that should be addressed first. For instance, each term has a summation over all the grid cells, $i = 0...S^2$, while most are also summarized over each anchor box, $j = 0...B$. Further, the λ parameters are constants which can be used for weighting, where by default: $\lambda_{coord} = 5$ and $\lambda_{noobj} = 0.5$. In addition, the discrete value $\mathbb{1}_{ij}^{obj}$ is defined as 1 if an object is present in the respective grid cell, while otherwise being 0, $\mathbb{1}_{ij}^{noobj}$ is the complement.

$$
\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \tag{2.13}
$$

The first term in the loss function is presented in equation 2.13 and is considered a localization loss. Where $x_i,\ y_i$ are coordinates of the ground truth bounding box centres, while $\hat{x}_i,\ \hat{y}_i$ are predicted values from the network. This means that the term is related to the position error, which is calculated, squared and added together for the two spatial dimensions.

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \tag{2.14}$$

The second loss term is linked to the size of the predicted bounding boxes and is shown in equation 2.14. Here $w_i,\ h_i$ are the width and height of the ground truth bounding boxes while $\hat{w}_i,\ \hat{h}_i$ indicate predicted values. The term is similar to the first, but differs in a square root on each width and height value. This is in order to partially address the fact that small deviations in large boxes should matter less than small deviations in small boxes.

$$\sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} (C_i - \hat{C}_i)^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{noobj} (C_i - \hat{C}_i)^2 \tag{2.15}$$

The third term calculates the loss associated with the confidence score for each predicted bounding box. It calculates intersection over union (see section 2.7.1), of the predicted bounding box with the ground truth, C_i, and subtracts the predicted confidence score, $\hat{C}_i$. This error term, as can be seen in equation 2.15, is further squared, and depending on whether there is an object present or not, is multiplied with or without the λ_{noobj} constant, respectively.

$$\sum_{i=0}^{S^2} \mathbb{1}_{i}^{noobj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2 \tag{2.16}$$

The fourth and last term of the loss function is displayed in equation 2.16 and is a sum of squared classification errors. Meaning $p_i(c)$ displays 1 for the correct class and 0 for all others, while $\hat{p}_i(c)$ is the predicted probability for each class. Since the discrete value $\mathbb{1}_{i}^{noobj}$ is included, the network will not be penalized for class predictions when there are no objects present.

Chapter 3

Data Collection

In order to train and validate the machine learning algorithm, there is a need for data. More specifically, several images of distinct objects with labels to their respective positions. For this master's thesis there is an interest in having images above and below water of the same object. A total of 582 images were generated above water and an additional 1209 images were generated below water. From this, three datasets are created, one above water, one below water and one hybrid, where the last is the previous two combined. In addition, a validation set with 169 images was extracted from the images. This chapter describes the setup and equipment used at the Marine Cybernetics Lab at NTNU Tyholt to produce these datasets.

3.1 Object

The object used in this report is a 12 oz. (0.34 kg.) hammer bought at the local hardware store. A hammer was chosen because of the characteristic shape it has and because a tool would be relevant for an AUV. The selected hammer is displayed in figure 3.1 and was picked due its distinct colors.

FIGURE 3.1: Tool Used In This Report[32]

3.2 Camera

There is a desire to minimize cost and therefore a simple and inexpensive camera is chosen for proof of concept. The device is a Raspberry Pi Camera Module V2 with 1920x1080 pixel resolution at 30 FPS[33]. For underwater purposes there is an added wide-angle lens, as can be seen in figure 3.2, which yields a 110 degree field-of-view. The assembly has a list price of $49 [34].

FIGURE 3.2: Raspberry Pi Camera Module V2 with Wide Angle Lens[34]

3.3 ROV Setup

The ROV used for this setup was a BlueROV2 with the heavy configuration, delivered from BlueRobotics and assembled at NTNU. It is a customizable robot with open-source electronics and software which makes it well suited for research. The current configuration can be seen in figure 3.3 and consists of eight thrusters which gives it high mobility in six degrees of freedom. Together with the camera module mentioned earlier, this setup comprises a lightweight, accessible and inexpensive way of adventuring underwater, with a list price of $5162 [35].

FIGURE 3.3: BlueROV2 Heavy Configuration[35]

3.4 Above Water Data Generation Setup

To ensure consistent image quality, the ROV and its camera was used above water. This meant manually rotating the ROV in the air using two researchers who aimed the 10kg vehicle towards the hammer as needed. An example of the starting position can be seen in figure 3.4, where the tool was laid out on a white background and the ROV rotated above in various angles. This was done for three different tool positions on the ground.

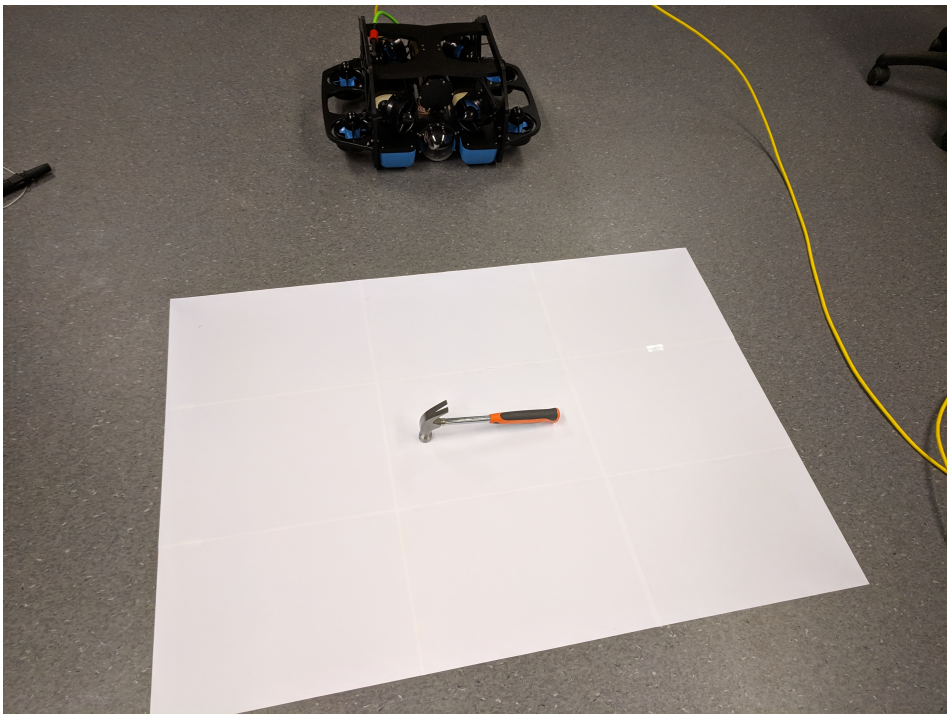


FIGURE 3.4: Single Hammer Setup

There was also a setup with two hammers together. This time they hovered above ground while the ROV was rotated around. Figure 3.5 shows how this looks like in a cluttered office environment. It can appear like the tools are hovering in midair, however they are attached with fishing lines to a suspended pole overhead.

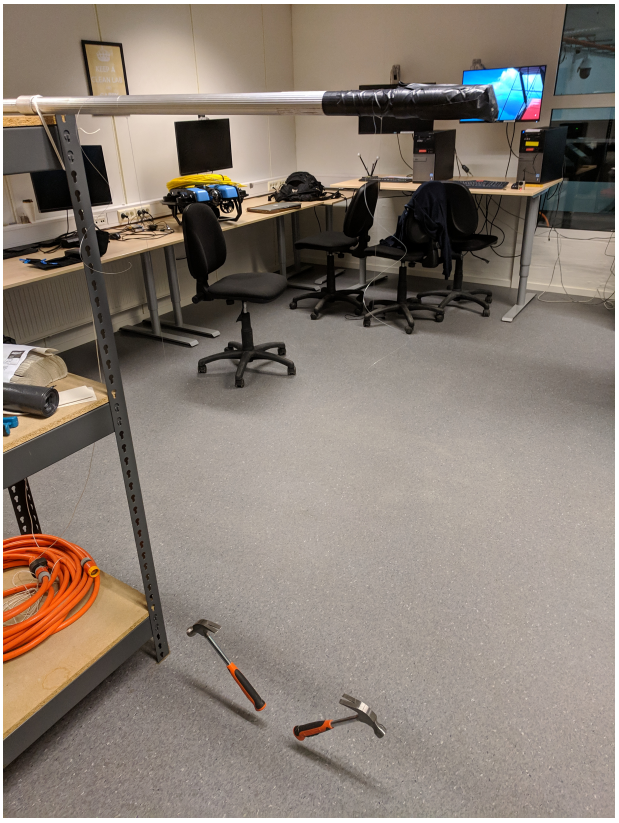


FIGURE 3.5: Dual Hammer Setup

3.5 Below Water Data Generation Setup

The images from below water were the most challenging to obtain since they required cooperation in different domains. A functioning ROV which responded and functioned as expected was critical. Issues like water penetration and code bugs were encountered and solved for during the time available to the team. Figure 3.6 shows the ROV primed and ready for action in the pool at the Marine Cybernetics Lab.

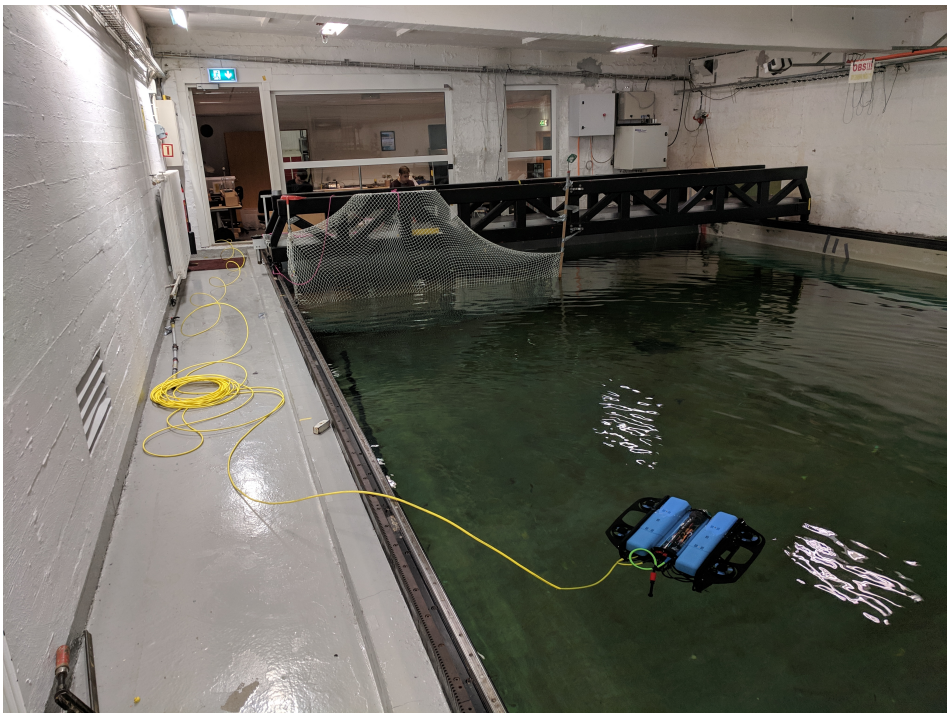
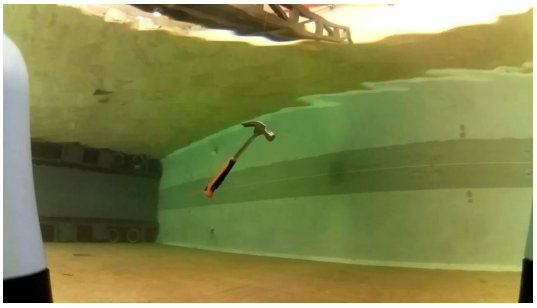


FIGURE 3.6: ROV Setup in Pool

The pole and fishing lines mentioned earlier were used again for the below water setup. Since the pool floor has a slight incline, the pole was fixed to a sliding carriage, which provided a straightforward way to control position and depth of the tools. In order to create a diverse dataset, the setup was repeated two times, one time with a single hammer and another time with dual hammers. To mitigate boyancy and to make sure they held the same position over time, a weight was added to the single hammer and the dual hammers were strung together at the shaft. Thus the ROV could rotate around to get a 360 degree view of each setup, as can be seen in figure 3.7 (a) and 3.7 (b) of the single and dual hammer setup, respectively. Note the wide-angle lens which captures a small portion of the ROV, especially on the left side.



(a) Single Hammer

(b) Dual Hammer

FIGURE 3.7: Underwater Images Taken From The ROV

3.6 Validation Dataset

To verify that the trained CNNs worked as expected, a slice of all the images was selected and used for validation. It is important that the validation dataset is representative of what one wants to detect, therefore a uniform distribution of all the images was sampled. Also, it is critical that the images used for validation are not part of the training set. Out of 1791 images and 2226 annotations, the validation set contained 169 images with 200 annotations. This means that 9.4% of the images and 9.0% of the annotations were reserved for validation.

3.7 Labels

In order to use images gathered through the setups described above, they need to be labeled and annotated. This is done with the tool LabelImg, which is an open-source software created by the Computer Science and Artificial Intelligence Laboratory at Massachusetts Institute of Technology (MIT)[36]. It lets the user manually draw bounding boxes and assign object labels while outputting the specific format style required to train YOLO. The program in action can be seen in figure 3.8 while example of an output file is located in appendix A.

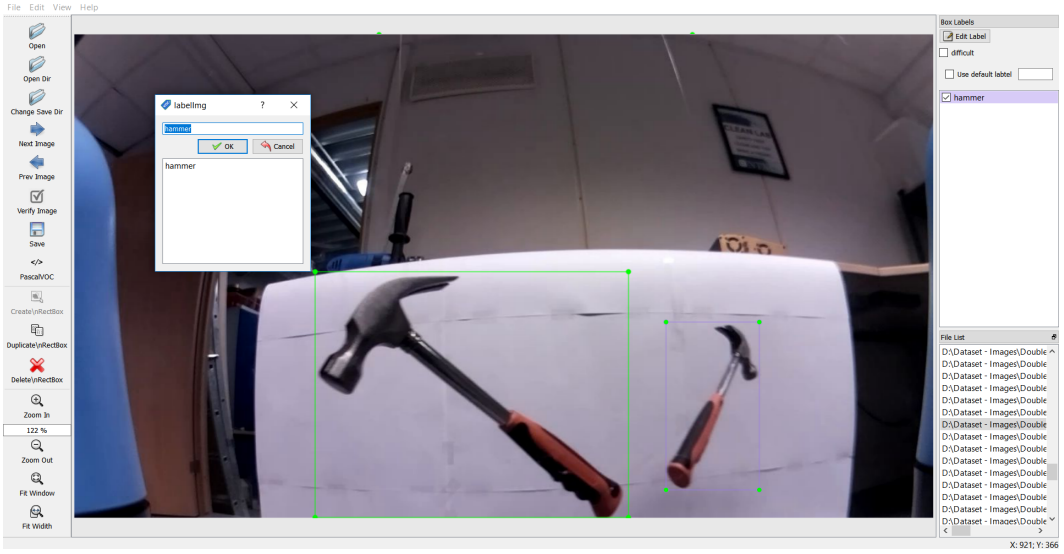


FIGURE 3.8: LabelImg Software[36]

Chapter 4

Implementation of CNN Method

When employing an algorithm created by someone else, it is seldom plug & play. There are several customizations and parameters which need to be applied, in addition there are translation issues between different programming languages and bugs that have to be dealt with. As mentioned in section 2.8, the convolutional neural network YOLOv2 was chosen for this report, where the following sections aim to explain some of the tools used to implement the tailored object detector and get it to function.

4.1 Python

One of the first things to decide when doing anything in machine learning is which programming language to write in. There are several possibilities, and the selection may just be a matter of personal preference. For this report however, the extensive user community that offered help and code snippets through forums, documentation etc. made it obvious to write in Python. Throughout the project this has proved to be extremely helpful, especially considering that the author had no previous experience coding the language.

4.2 Google Colaboratory

All of the images, annotations and results used in this report, together with the thesis itself, can be found in the following link (*https://drive.google.com/drive/folders/1O0hdMYZwifGeHSnysH_xl4ynkTPovILz?usp=sharing*), through the cloud service Google Colaboratory[37], also known as Google Colab. It was released to the public in October 2017 and is a tool which lets users run Python code directly in their browser[38]. This

is achieved by using the existing feature Google Drive for storage and the open source project Jupyter Notebook as a user interface. In addition, they offer free access to a server farm of GPUs, namely the NVIDIA Tesla K80 graphics cards, as seen in figure 4.1. Being top-of-the-line in 2014, these graphic cards have 4992 cores each and are rated for a maximum throughput of 2.9 tera floating point operations (teraflops)[39]. This allows users to vastly increase the number of calculations per unit time. Although convenient at runtime, it is especially useful during training of the network, when the algorithm calculates several thousand matrix multiplications per image.

FIGURE 4.1: Tesla K80 Graphics Card[39]

4.3 Tensorflow

When Google announced that they would release their internal machine learning library TensorFlow to the public in November 2015, it marked a critical milestone in the development of deep learning research. With support for Python, C++ and CUDA, they opened up a world where the common person could learn from, and contribute to, the deep learning community[40]. It is a library that has quickly gained ground, thus note their logo which is displayed in figure 4.2. TensorFlow consists of several high-level functions that together with other libraries like Keras, make it very simple to define any type of layer in a neural network architecture. Something which would require several pages of tightly packed code could instead be done on a single line. The functions have parameters like how many neurons to use, what size and how many convolution kernels to apply, what type of activation function to use and whether to do pooling on a layer.

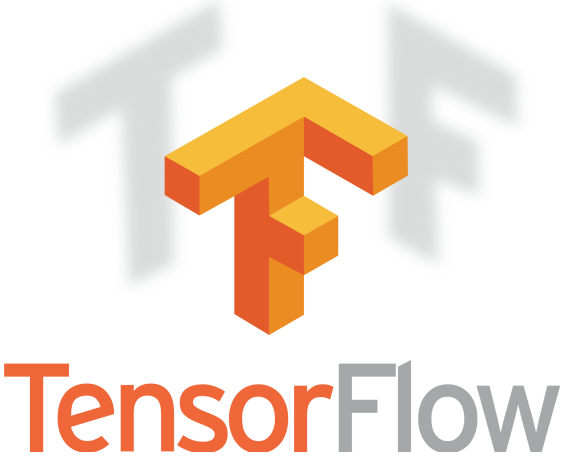


FIGURE 4.2: TensorFlow Machine Learning Library Logo[40]

4.4 Darkflow

The YOLO architecture known as Darknet was originally published by Joseph Redmon et al. in their first paper[17] and the code was made freely accessible through the authors website (*https://pjreddie.com/darknet/yolo/*). A challenge with this code is that it was written in C, something which might be advantageous if one wants to run the program on a microchip. For this report however, an implementation in Python is preferred, since that makes the free GPU services from Google Colab available. Luckily there is a staff member at Google, Trieu H. Trinh, which made this possible through a library known as Darkflow[41]. The repository is a translation from the original YOLO architecture Darknet, written in C, to the TensorFlow library written in Python, hence Darkflow.

4.5 Building the Neural Network

Even though many of the libraries are openly available, there are several modifications needed in order to build them on any machine. Obviously this includes the successful installation and running of Python, Tensorflow and Darkflow, as well as the required dependencies. In addition, there are also specific considerations for the local machine. Something which might take a long time for beginners, is working out the folder structure and to keep track of working directories. Different programs need to run from their distinct place and configuration files are required to match. For example, the original detector is designed to predict on several different classes, but now the configuration files and layers are changed to only match one output, "hammer". Further, there are hyperparameters that can be tuned, like how many epochs to run, learning rate and what batch size to give the GPU.

4.6 Training the Neural Network

After building the YOLO architecture and verifying that it runs as expected, the next step is to train on a custom dataset. Although the network will train on completely new classes, it will not start from scratch. The authors of YOLO published already trained weights from the Microsoft dataset competition Common Objects in Context (COCO)[42], together with their code. These weights are written in the C programming language, which meant they needed translation to match with Darkflow in order to be used as a starting point.

The hyperparameters mentioned in the previous section affect how training is executed. Batch size defines how many images are sent to the GPU per iteration, since it is impractical to use all the training data at once, mainly because of limited memory inside the GPU. For this report a batch size of 8 was found to work fine. After each iteration, the neural network weights are changed by a number known as learning rate. This controls how aggressively the detector learns and is typically a number between 0.01 and 0.0001. For the YOLOv2 algorithm this number is 0.01. When the network has iterated through all the images given, it is said to complete one epoch. The detector in this report ran 50 epochs per dataset.

4.7 Validating the Neural Network

After a neural network has finished training, it is time to validate. The importance of using an unbiased method is especially present when there are several object detectors to compare. In this thesis, all parameters are kept equal except the three different input datasets. A validation set was created, see section 3.6, and each trained detector generates predictions on this. Including bounding box size with coordinates, confidence score and label. Note that only predictions with confidence score above a specified threshold value are considered, as is common practice in object detection. For this report a threshold of 0.3 is chosen. The predictions are then compared with ground truth annotations to calculate a Precision x Recall curve and integrated to find the Average Precision. This is done with the help of code from Rafael Padilla, a PhD candidate in machine learning at Universidade Federal do Rio de Janeiro[43]. His code requires a specific format, therefore a translation script was written for the predictions and ground truth annotations.

Chapter 5

Results

In this chapter, the results gathered from each separate object detector is presented. Since the neural networks are based on three separate datasets, there are three different detectors with their own characteristics. In order to compare the outcomes, they were validated on the same benchmark dataset of 169 images. The results from this validation are further explained in the next sections, in addition to a few images which demonstrate the strengths and weaknesses of each detector. However, before examining these results, the time it took to manually label all the images is presented and examined for future reference.

5.1 Time Spent Manually Labelling Images

Manually labelling large amounts of images can be quite time consuming, but is essential in order to have a functioning machine learning model. Labelling is often dreaded by researchers and graduate students alike and it is therefore of interest to have a sense of how much time is spent in this realm. This is presented for each of the datasets below. As per chapter 3, there are three different datasets: above water, below water and hybrid, which is a combination of the previous two. Note that the time spent is independent of the validation images since they were extracted after the labelling process.

5.1.1 Above Water Dataset

The images above water were separated into five different sections, depending on how they were captured, in order to make them more manageable. The results for the time taken to label these sections are summarized in table 5.1, with the number of images, number of annotations plus the total time taken. In addition, there are two calculated

Section	Images	Annotations	Total Time [mm:ss]	$\frac{Time}{Image}$ [s]	$\frac{Time}{Annotation}$ [s]
1	160	319	31:41	11.9	5.96
2	132	264	25:15	11.5	5.74
3	140	140	13:51	5.94	5.94
4	102	102	10:00	5.88	5.88
5	91	91	7:37	5.02	5.02
Total	625	916	88:24	8.49	5.79

TABLE 5.1: Manual Labelling Time for Above Water Dataset

values: time per image and time per annotation. This is to highlight that some sections had two objects in almost all of the images, while others had only one, see chapter 3. In total the dataset took 88 minutes, almost one and a half hour to label, with a total of 625 images and 916 annotations. The time spent per image averaged 8.49s in total, while the time spent per annotation averaged 5.79s.

5.1.2 Below Water Dataset

Section	Images	Annotations	Total Time [mm:ss]	$\frac{Time}{Image}$ [s]	$\frac{Time}{Annotation}$ [s]
6	84	166	26:43	19.1	9.66
7	60	119	13:35	13.6	6.85
8	35	68	8:10	14.0	7.21
9	346	346	38:56	6.75	6.75
10	128	128	14:09	6.63	6.63
11	138	138	16:14	7.06	7.06
12	129	129	17:44	8.25	8.25
13	260	260	30:29	7.03	7.03
14	154	154	15:48	6.16	6.16
Total	1335	1509	181:48	8.17	7.23

TABLE 5.2: Manual Labelling Time for Below Water Dataset

The below water dataset is a little bit larger than the above water dataset, and is shown in table 5.2 with the same format. The dataset consists of nine sections, 1335 images and 1509 annotations. In total it took 182 minutes, a little over three hours, to label it all. This gave an average time per image of 8.17s and an average time per annotation of 7.23s.

5.1.3 Hybrid Dataset

Images	Annotations	Total Time [mm:ss]	$\frac{Time}{Image}$ [s]	$\frac{Time}{Annotation}$ [s]
1960	2425	270:12	8.27	6.69

Table 5.3: Manual Labelling Time for Hybrid Dataset

The hybrid dataset results, which in essence are all images combined together, are summarized in table 5.3. Where the total time spent by the author amounts to 270 minutes, or four and a half hours, to label 1960 images with 2425 annotations. As mentioned previously, there are several images which have two hammers in them, resulting in more than one annotation per image. In fact, there are 23.7% more annotations than images, and consequently 23.7% more time spent per image than per annotation.

5.2 Object Detector Validation

The following sections present evaluation parameters for each detector, respectively. Where impartial parameters are used in order to compare different object detectors. Evaluation metrics are previously described in section 2.7, where the Average Precision is found by integrating the Precision x Recall curve. Remember that a result of 100% AP means that all objects were found and no false predictions were made. Each detector was evaluated on the same validation dataset, which as outlined in section 3.6, contains 169 images and 200 annotations.

5.2.1 Object Detector Trained on Above Water Dataset

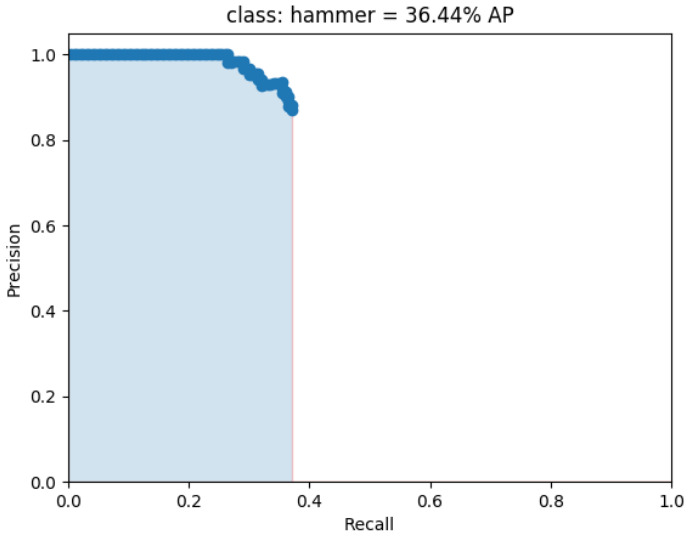


FIGURE 5.1: Precision x Recall Curve and Average Precision for Above Water Dataset

Figure 5.1 shows the Precision x Recall curve and the Average Precision of the object detector which was trained on images solely from above water. This dataset is the smallest of all three, but took almost two hours to train on the NVIDIA Tesla K80 graphics card, from Google Colab's servers, see section 4.2. Of the 200 ground-truth annotations, it managed to predict 74 true positives (correct predictions), 11 false positives (false predictions) and 126 false negatives (missing predictions). By integrating the Precision x Recall curve the average precision was found to be 36.44%.

5.2.2 Object Detector Trained on Below Water Dataset

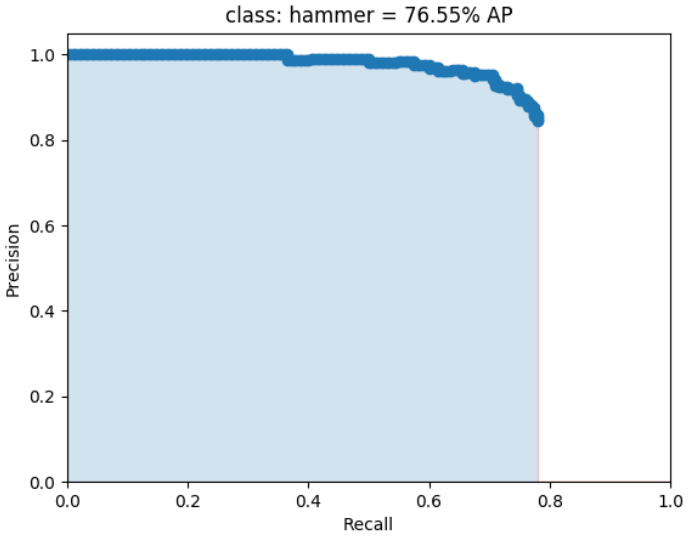


FIGURE 5.2: Precision x Recall Curve and Average Precision for Below Water Dataset

The object detector trained on the below water dataset had 156 true positives, 29 false positives and 44 false negatives on the validation set. This resulted in an Average Precision of 76.55%, and a Precision x Recall curve as can be seen in figure 5.2. The detector spent a little over three hours to train 1209 images with 1366 annotations on Google Colab.

5.2.3 Object Detector Trained on Hybrid Dataset

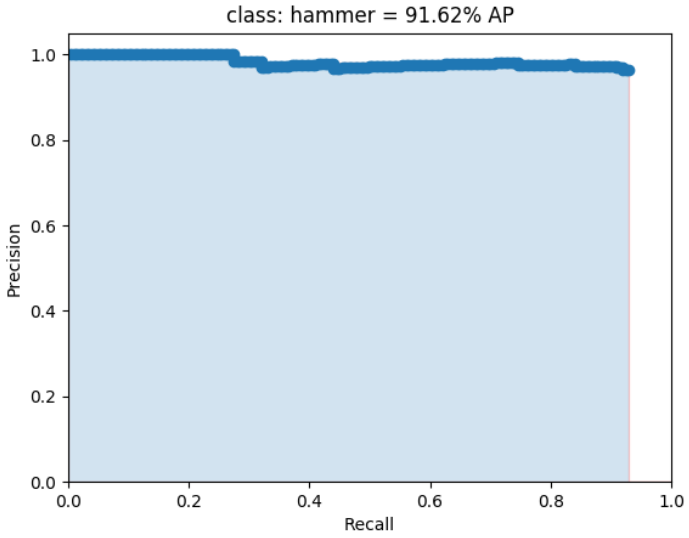


FIGURE 5.3: Precision x Recall Curve and Average Precision for Hybrid Dataset

The object detector trained on the hybrid dataset, which is a combination of the images in the two other datasets, took three and a half hours to train on Google Colab. It boasts in total of 1791 images, 2226 annotations and resulted in a prediction of 186 true positives, 7 false positives and 14 false negatives. These values were used to create the Precision x Recall curve, as can be seen in figure 5.1, and to calculate the Average Precision, which was 91.62%.

5.3 Example Images

To better understand how each of the three detectors work, and get a feel of what the differences look like, a couple of images are presented below. They were selected to give a representation of the strengths and weaknesses of each detector. There are four different parts, where the first compares each neural network to itself. The second points out how demanding it can be to fulfill the strict evaluation criteria, set by the Pascal VOC Challenge, while the third compares the detectors to each other. Finally a subset of images which had objects that were deemed challenging to predict are displayed.

5.3.1 Similar Images Yield Different Results

All the detectors have examples of inconsistencies amidst themselves. Where two images that appear to be similar, return different results.

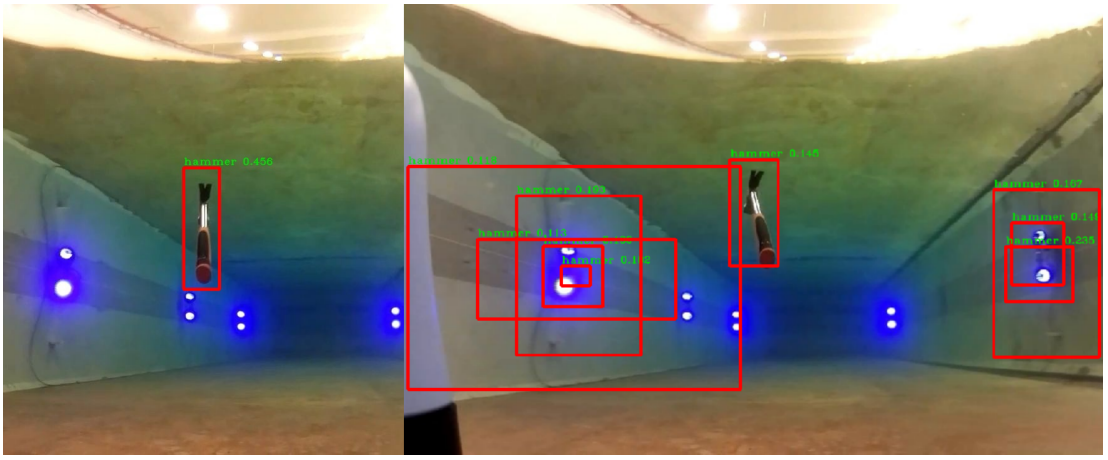


FIGURE 5.4: Similar Images Produce Different Results by Above Water Detector, Left: Correctly Detected, Right: Similar Image Displaying All Predictions

When considering the object detector which has been trained on above water images only, it is proper to use an example which arises from the opposite subset, namely below water. Figure 5.4 shows two images which are very similar in background and shape, but yield different detections. On the left side a hammer is recognized with a confidence of 0.456, while on the right side there are no legitimate predictions. However, by removing the threshold value which requires confidence score to be larger than 0.3, see section 4.7, it is possible to check how far off the detector really was. For example, there was one prediction from the right image which is perfectly aligned with the object, but that had a confidence score of 0.145, thus it did not count. In addition, there were also eight other predictions which were nowhere near, ranging in confidence from 0.100-0.235. Similarly bad predictions were made on the left image also, with confidence score between 0.108-0.217, but they are not shown here.

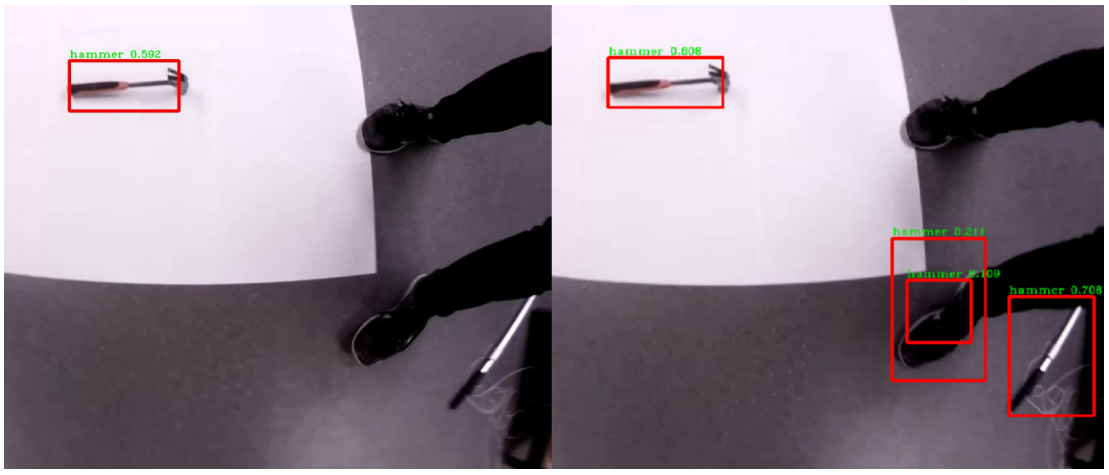


FIGURE 5.5: Similar Images Produce Different Results by Below Water Detector, Left: Correctly Detected, Right: Similar Image Displaying All Predictions

When examining the validation images from the neural network which was trained on the below water dataset, there are similar findings. Two images from the opposite subset, namely above water, is shown in figure 5.5. Where the image on the left recognizes a hammer correctly with confidence of 0.592, while the image on the right recognizes the same hammer with a confidence score of 0.608. In addition, it predicts a false positive hammer with confidence of 0.708 in the lower right hand corner. By going below the threshold confidence score value of 0.3, it is possible to see two other predictions which were nowhere near the tool, ranging in confidence scores from 0.109-0.211. These three false predictions were also made by the left image, with confidence scores ranging from 0.106-0.276, but are not shown here, and since the scores are lower than 0.3, they are not considered legitimate predictions.

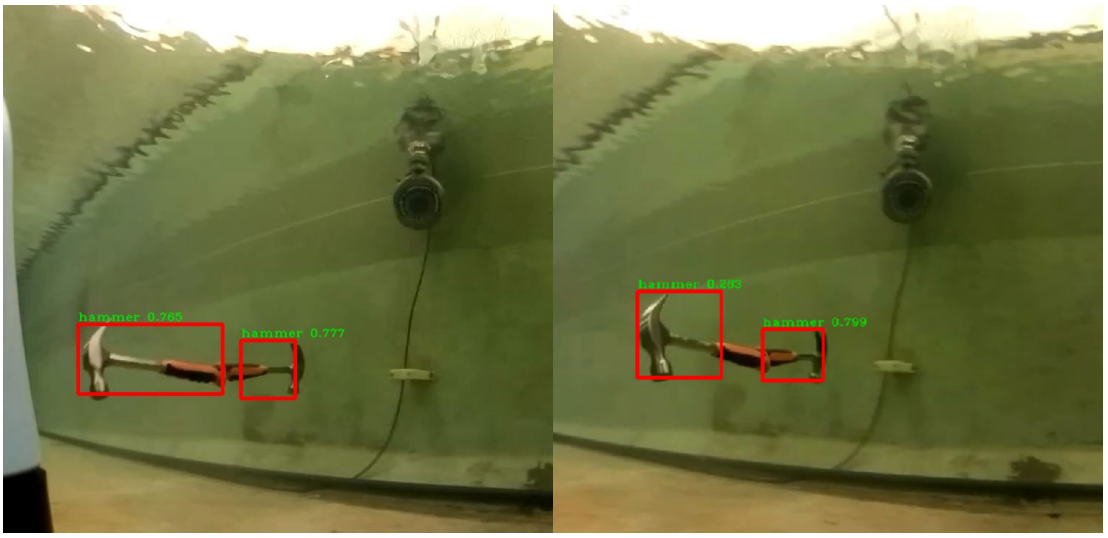


FIGURE 5.6: Similar Images Produce Different Results by Hybrid Detector, Left: Correctly Detected, Right: Similar Image Displaying All Predictions

This tendency is also found when the hybrid dataset was used to train a detector. Two images which are different by a couple of frames are displayed in figure 5.6. On the left side, two hammers are recognized with a confidence score of 0.765 and 0.777. While on the right side, only one of two hammers is considered a true prediction with a confidence of 0.799. By delving deeper it is seen that the right image also made a prediction on the second hammer, but with a confidence score of 0.263, it was not counted as a detection. However, with such an inaccurate bounding box it would probably be counted as a false negative if included. The left image had no other predictions.

5.3.2 IOU Too Strict

The Pascal VOC Challenge requires that IOU be greater than 0.5 in order to count as a correct prediction, see section 2.7. In some cases, especially for small objects, this can be demanding to obtain. An example is demonstrated in figure 5.7, where the same image is presented, but from different object detectors, together with ground truth on the right. The left image is from using the hybrid dataset, and predicts a hammer with confidence of 0.845, while the middle image used the below water dataset and predicted a hammer with 0.932 in confidence. However, they were both counted as false positives by the evaluation method because of insufficient overlap between their labels and ground truth, displayed on the right for comparison.

FIGURE 5.7: IOU Might be Too Strict, Left: Hybrid, Middle: Below Water, Right: Ground Truth

A similar situation is illustrated in figure 5.8, where one image with predictions from each of the three object detectors, together with ground truth, is shown. The leftmost image is from the hybrid detector and displays a prediction with confidence of 0.796. However, it was counted as a false positive because of inadequate overlap with the ground truth. The two middle images are from the below water and the above water detector, respectively, and have predictions with 0.764 and 0.607 confidence score. Both of these predictions were considered true positives by the Pascal VOC Challenge. When visually comparing to the ground truth, shown on the right, and the false positive prediction by the hybrid detector, on the left, this demonstrates how meticulous the evaluation metrics can be.

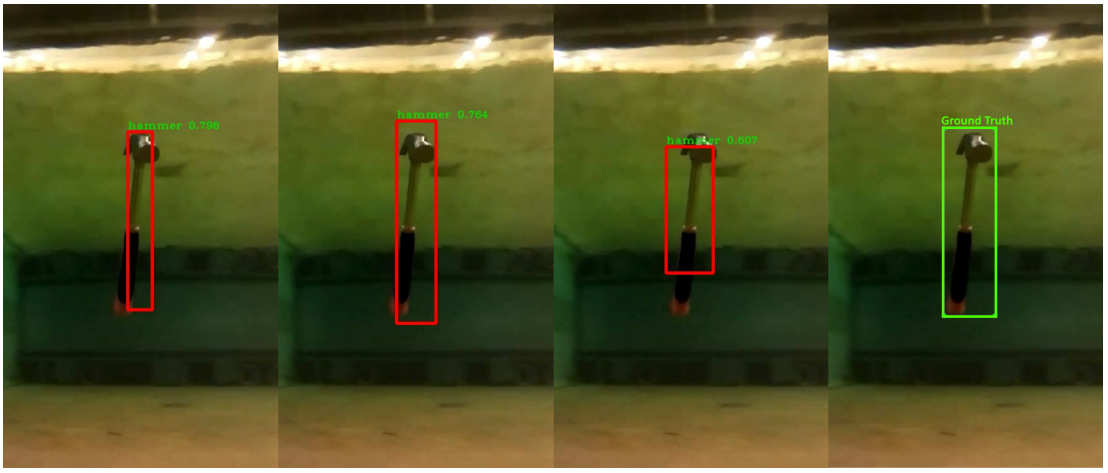


FIGURE 5.8: IOU Might be Too Strict, Left: Hybrid, Middle-Left: Below Water, Middle-Right: Above Water, Right: Ground Truth

5.3.3 Images Only Found by Lower Detectors

After obtaining results from running the validation metrics, as described in section 5.2, it was established that the neural network which was trained on the hybrid dataset had highest Average Precision. It would therefore be logical to assume that this detector outperformed the two other in every aspect, but indeed there are several instances of object detectors with lower Average Precision score outperforming their counterparts. For example in the aforementioned figure 5.8, where the hybrid detector was the only one which did not recognize the object.

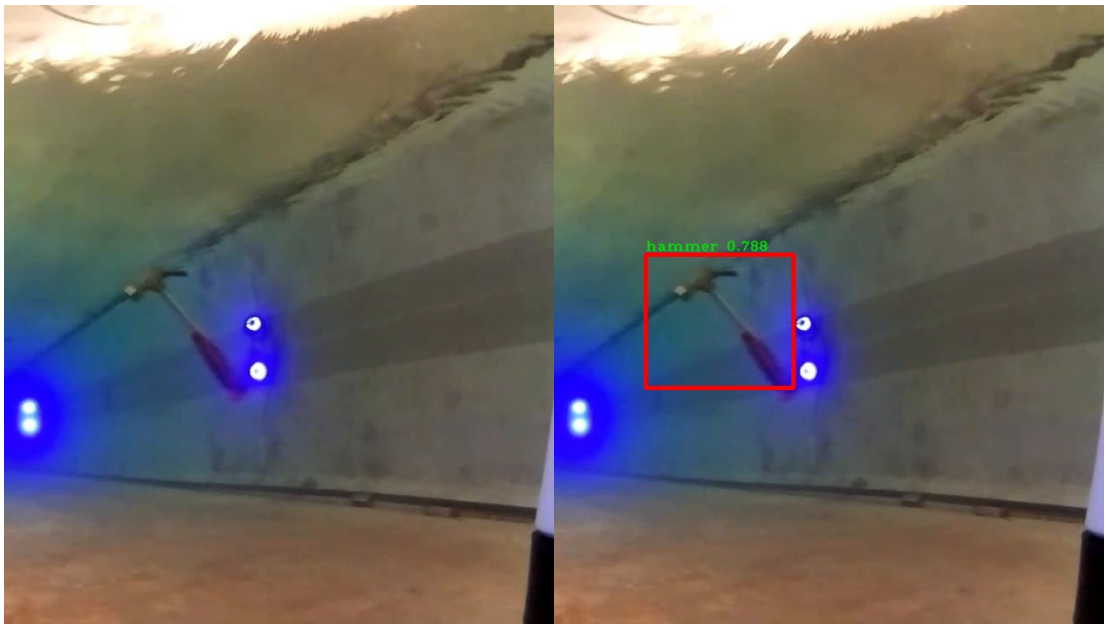


FIGURE 5.9: Object Only Found by Below Water Dataset, Left: Hybrid, Right: Below Water

This can also be seen in figure 5.9, which shows the same image after two different object detectors have made predictions. The right side, which is from the below water detector and had a lower AP score, detected a hammer with 0.788 in confidence score. Left side however, which is from the hybrid detector, did not detect any object. In fact, by checking for predictions below 0.3 in confidence, it was asserted that the hybrid detector actually made zero predictions on this image.

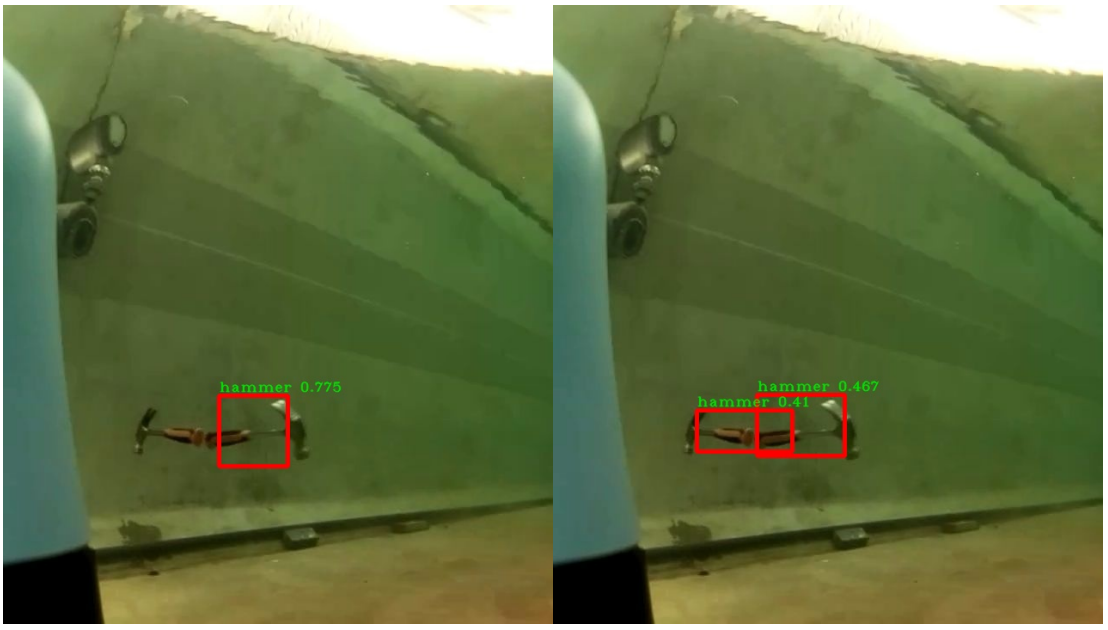


FIGURE 5.10: Object Only Found by Below Water Dataset, Left: Hybrid Dataset, Right: Below Water Dataset

Another example is presented in figure 5.10, where again, the lower performing detector trained on below water dataset (right image) recognizes more hammers than the detector trained on hybrid dataset (left image). By analyzing the output data, it is possible to see that the hybrid detector actually made two predictions which were not accepted. One with 0.290 in confidence which would have been a true positive if included, and another with 0.187 in confidence which was off target and would have been a false positive.

5.3.4 Difficult Images

Of the 169 images in the validation dataset, some are considered more challenging than others. A selection of three examples are shown in figure 5.11, and the images are displayed after they were predicted on by the detector which was trained on the hybrid dataset. Left image shows a hand picking up a hammer in action, where the detector managed to recognize a hammer with confidence score of 0.884. The middle image shows part of a hammer out of focus from a weird angle, which none of the object detectors managed to predict accurately, and on the right side is a blurry image with disturbances from underwater light sources. Nonetheless, the detector prevailed and managed to recognize a hammer with 0.748 in confidence score.

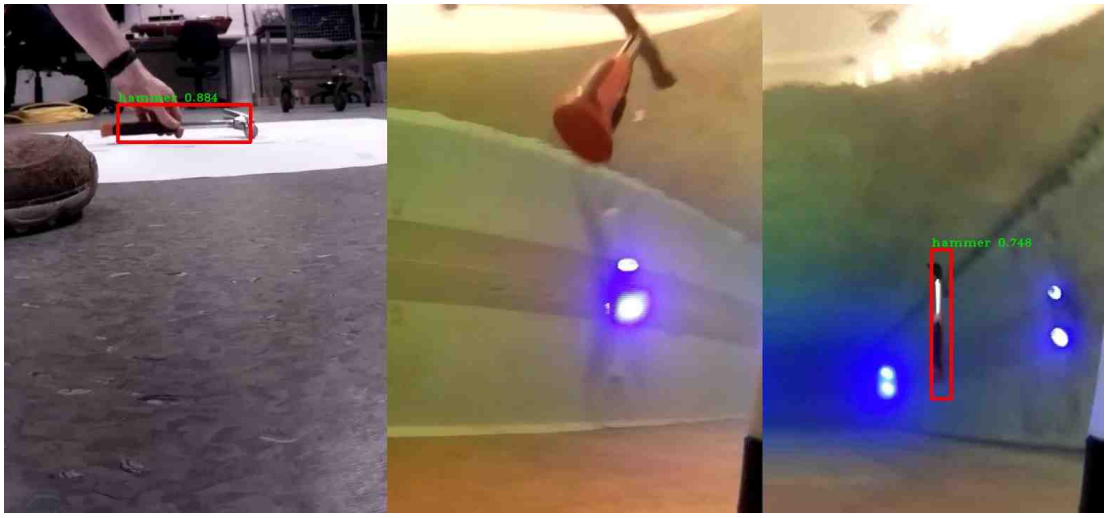


FIGURE 5.11: Example Images from the Validation Dataset

Chapter 6

Discussion

6.1 Time Spent Manually Labelling Images

The manual labelling time which was presented in section 5.1, gives insight into how long it took to create a dataset for this thesis. Hopefully it can serve as a rough estimate for other researchers who want to collect their own data. However, it should be noted that the total time averages of 8.27s per image and 7.23s per annotation might not be applicable everywhere. The results are valid for the labelling procedure only, and does not contain any information regarding the actual collection of images using an ROV or the setup in the lab. Neither does it incorporate the time it took to filter or sort the raw data. Such externalities will differ depending on what arrangements are used, and thus affect the time budget accordingly.

6.2 Object Detector Validation

When it comes to validation of the datasets, this was done with evaluation metrics from the Pascal VOC competition and a fixed validation dataset. This resulted in three different Average Precision scores of 36%, 77% and 92% for each of the object detectors. By looking at the underlying datasets, one would expect such a distribution just from the number of images available to each neural network, 582, 1209 and 1791, respectively. However when comparing the detectors, an acceptable review would ask how one could be certain that they are finished training. This report used fixed hyperparameters, with 50 epochs, meaning that each neural network ran through all the images 50 times, independent of how long time it took. The results show that the time it took to train varied by almost 100% from the smallest to the largest dataset. Thus, to assess whether this was a sufficient amount of training, the loss function output, as described in section

2.8.2, was analyzed. For all the neural networks, it reduced gradually from a maximum value around three hundred to below one in around 15 epochs. After that, it stagnated and oscillated between zero and one (remember that it can only take on non-negative values). Although a variation of the hyperparameters would be interesting to document, the loss function provides support for the argument that there is little to gain from extra training.

6.3 Similar Images Yield Different Results

Even though one can expect the training to be complete, it is not certain that the results are adequate in order to function as subsea object detectors. The examples which are provided in section 5.3.1, show instances where the neural networks fail to detect a tool, while they are fully capable in a slightly different situation. This might be sign of an unreliable detector, which needs close to "perfect" conditions for a correct prediction. However, since there are several predictions made that have confidence below the 0.3 threshold, this suggests that the detectors are on the right track and more training data would push it to the higher end of reliance and Average Precision scores. This is supported by the variety of angles that have indeed been predicted correctly. In addition, there are a few images in the validation dataset that seem truly difficult, where it might not be reasonable to expect proper predictions from any object detector.

6.4 IOU Too Strict

Some of the images presented in the results, especially in section 5.3.2, pointed out that the IOU metric used as a basis for calculating Average Precision, as defined by the Pascal VOC Challenge, might make it difficult to obtain true positive predictions. Since it demands an overlap of 0.5 between predicted and ground truth label, it is more sensitive to inaccuracies in labelled data for smaller objects. The question then becomes: is this too strict? Take the two hammers in figure 5.7 as an example, both of them are considered to be false negative predictions. This is not necessarily obvious to the naked eye without a detailed calculation of the IOU for each image. If the prediction is fair enough for a human, does it really matter that the IOU has to be above 0.5 at all times?

An answer to that would highly depend on which task the detector was performing. If an AUV in a subsea environment was trying to grab the object, for example. One could argue that it would be more important to have a general idea of where the hammer was at a distance, instead of a false negative classification which potentially could cause more

confusion. When the AUV moves closer to the tool, it would be relatively larger in the camera frame and therefore easier to predict by the network. Shortcomings of the Pascal VOC competition metrics have also been addressed previously by Joseph Redmon et al. in the latest YOLO paper[19].

6.5 Images Only Found by Lower Detectors

Another point was made regarding the fact that the detector with highest Average Precision did not manage to detect all objects that the lower scoring detectors did. One would expect the detector based on the hybrid dataset to be at least as good as the two lesser performing neural networks, since it has been trained on the combination of their datasets. However, as has been demonstrated in section 5.3.3, there are examples of detectors with lower Average Precision outperforming their counterparts.

In trying to understand why this is the case, it might be of interest to know that every time a neural network is finished training, it is very common to end up with slightly different results. This is due to the fact that the training dataset runs in batches instead of all at once, see section 4.5, and that this batch is randomly seeded. Therefore, it is not improbable that these small deviations may account for the observed differences. Even though they are between neural networks which have been trained on the same subset of images.

In addition, differences between neural networks which have been trained on varying number of images, raise the topic of overfitting. This is when a network learns the detail and noise in the training data to the extent that it negatively impacts performance of the model on new data. As such, the hybrid detector might have picked up some cues that make it difficult to predict on unseen images. It seems unlikely however, since the added images are from a different domain (above vs. below water). Also, the selected examples, where applicable, are purposefully chosen from the opposite domain than the training data used for that neural network. Consequently, a fairer criticism is whether the neural network filters might become confused because of the different domains they have to accommodate.

Chapter 7

Conclusions and Further Work

7.1 Conclusions

The main objective of this thesis was to implement and train a deep learning object detector to recognize images under water. This is an important topic for offshore installation owners and operators, as it is one step closer to fully autonomous underwater operations. This can give huge economic savings and ensure higher operability.

A basic introduction to some of the fundamental principles of deep neural networks was introduced. Afterwards YOLOv2 was chosen as the viable object detector, and implemented in Python using cloud computing from Google. The object detector was trained on three different image datasets: above water, below water and hybrid (a combination of the other two), which were gathered from scratch in the Marine Cybernetics lab at Tyholt NTNU, using the BlueROV2, and its camera. These images were in turn manually labelled and used to train the selected algorithm. A fixed dataset, which includes images from both above and below water, totalling 9.2% of all images collected, was used for validation. This resulted in Average Precision scores of 36%, 77%, and 92%, respectively.

Thus, a real-time object detector which is able to predict on underwater images is presented. With Average Precision scores showing that it is possible to customize a convolutional neural network so that it can detect any object underwater, given relevant training data. Although it is necessary to note, that the images presented in this report have ample lighting and clear water, something which might not reflect actual subsea conditions. However, there are substantial examples which indicate that this could be solved by training on a greater amount of images from the relevant domain. Even images from above water, which can be cheaper to come by, would increase precision scores.

7.2 Further Work

Further work would take the object detectors from this report and apply them in a real-world subsea scenario on board an ROV computer so that they can be tested in real-time on limited hardware. Such a test would provide great insight into how robust the detectors are and how they handle subsea seawater, as compared to the clear laboratory environment. This opens the door for other applications like navigation and station keeping based on object detection.

Generally, when it comes to machine learning, it is acknowledged that a larger dataset returns better results. It is therefore suggested, to train on a higher number of images with more variance, to gain a robust object detector. One way to create new images is by augmenting the previously collected data. This is done through automated processes like random crop, flip, distort, random resize and added effects like saturation, hue and jitter. Such data augmentation techniques can increase the original dataset by several multiples, thereby yielding a detector with higher invariance towards noise, translation etc.

An evaluation method is appreciated for its conformity to an objective, thus it is desirable that a high validation score represents substantial accordance with the set goals. As mentioned in the discussion, there might be discrepancies between the Pascal VOC competition metrics and the goals of an object detector. Thus a refined evaluation metric which takes this into account, is sought after in future endeavours.

When it comes to the validation dataset, there might be a concern of systematic biases. Since it consists of a uniform distribution totalling 9.4% of all the images, there are more images from the below water dataset, than above water dataset. This in turn means that there could be a disproportionate advantage given to the below water dataset. A future work would take this into consideration and select a validation method which reduces biases, cross-validation for example.

Appendix A

Label Output Format

```
<?xml version="1.0"?>
<annotation>
    <folder>Experiment 2 - Two Hammers White Backgroung With Rotation</folder>
    <filename>ex2 (42).jpg</filename>
    <path>D:\Dropbox\NTNU\10. Semester\Dataset - Images\Labelling\Experiment 2 - Two
        Hammers White Backgroung With Rotation\ex2 (42).jpg</path>
  - <source>
        <database>Unknown</database>
    </source>
  - <size>
        <width>1280</width>
        <height>720</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
  - <object>
        <name>hammer</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
      - <bndbox>
            <xmin>639</xmin>
            <ymin>41</ymin>
            <xmax>977</xmax>
            <ymax>310</ymax>
        </bndbox>
    </object>
  - <object>
        <name>hammer</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
      - <bndbox>
            <xmin>203</xmin>
            <ymin>90</ymin>
            <xmax>531</xmax>
            <ymax>403</ymax>
        </bndbox>
    </object>
</annotation>
```

Bibliography

[1] *"Eelume"*. Accessed 15.11.2018. URL: https://eelume.com.

[2] Esten Ingar Grøtli et al. "Towards more autonomous ROV operations: Scalable and modular localization with experiment data". In: *IFAC-PapersOnLine* 49.23 (2016), pp. 173–180. ISSN: 24058963. DOI: 10.1016/j.ifacol.2016.10.339.

[3] *"Uncontended Voice and Data Through TDMA Networks, Marine Technology Reporter"*. 2016. URL: https://www.marinetechnologynews.com/news/uncontended-voice-through-networks-543067.

[4] Eirik F Kjærnli. *"Deep Reinforcement Learning Based Controllers In Underwater Robotics"*. 2018.

[5] Abir Khan. *"Deep Reinforcement Learning based tracking behavior for underwater vehicles"*. 2018.

[6] Yali Amit and Pedro Felzenszwalb. *"Object Detection Definition"*. 2014. URL: http://static.cs.brown.edu/people/pff/papers/detection.pdf.

[7] Y. LeCun et al. *"Backpropagation Applied to Handwritten Zip Code Recognition"*. 1989. DOI: 10.1162/neco.1989.1.4.541.

[8] 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* 1 (2001), pp. I–511–I–518. ISSN: 1063-6919. DOI: 10.1109/CVPR.2001.990517. arXiv: arXiv:1011.1669v3. URL: http://ieeexplore.ieee.org/document/990517/.

[9] N. Dalal and B. Triggs. "Histograms of oriented gradients for human detection". In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 1. June 2005, 886–893 vol. 1. DOI: 10.1109/CVPR.2005.177.

[10] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision* 115.3 (2015), pp. 211–252. ISSN: 15731405. DOI: 10.1007/s11263-015-0816-y. arXiv: 1409.0575. URL: http://dx.doi.org/10.1007/s11263-015-0816-y.

[11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Neural Information Processing Systems* 25 (Jan. 2012). DOI: 10.1145/3065386.

[12] Mark Everingham et al. "The pascal visual object classes (VOC) challenge". In: *International Journal of Computer Vision* 88.2 (2010), pp. 303–338. ISSN: 09205691. DOI: 10.1007/s11263-009-0275-4. arXiv: 1411.4038.

[13] Ross B. Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *CoRR* 1311.2524 (2013). arXiv: 1311.2524. URL: http://arxiv.org/abs/1311.2524.

[14] Ross Girshick. "Fast R-CNN". In: *International Conference on Computer Vision* (2015), pp. 1440–1448. ISSN: 15505499. DOI: 10.1109/iccv.2015.169. arXiv: 1504.08083.

[15] Shaoqing Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *Journal of Acquired Immune Deficiency Syndromes* (2015), pp. 1–9. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2016.2577031. arXiv: 1506.01497.

[16] Kaiming He et al. "Mask R-CNN". In: *Proceedings of the IEEE International Conference on Computer Vision* 2017-Octob (2017), pp. 2980–2988. ISSN: 15505499. DOI: 10.1109/ICCV.2017.322. arXiv: 1703.06870.

[17] Joseph Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". In: *CoRR* abs/1506.02640 (2015). arXiv: 1506.02640. URL: http://arxiv.org/abs/1506.02640.

[18] Joseph Redmon and Ali Farhadi. "YOLO9000: Better, Faster, Stronger". In: *CoRR* abs/1612.08242 (2016). arXiv: 1612.08242. URL: http://arxiv.org/abs/1612.08242.

[19] Joseph Redmon and Ali Farhadi. "YOLOv3: An Incremental Improvement". In: *CoRR* abs/1804.02767 (2018). arXiv: 1804.02767. URL: http://arxiv.org/abs/1804.02767.

[20] Jason Parham et al. "Animal Population Censusing at Scale with Citizen Science and Photographic Identification". In: *Association for the Advancement of Artificial Intelligence* (2017), pp. 37–44. ISSN: 0956053X. DOI: 10.1016/j.wasman.2010.12.019.

[21] Florian Shkurti et al. "Underwater multi-robot convoying using visual tracking by detection". In: *IEEE International Conference on Intelligent Robots and Systems* 2017-Septe (2017), pp. 4189–4196. ISSN: 21530866. DOI: 10.1109/IROS.2017.8206280. arXiv: 1709.08292.

[22] Mikkel Cornelius Nielsen. *"Machine Learning for Underwater Robotics"*. URL: https://drive.google.com/drive/folders/1mjRAmylDlBMiQEQggpRCBRBc4k9MIVQV.

[23] Ramon Quiza and J. Paulo Davim. "Computational Methods and Optimization". eng. In: *Machining of Hard Materials*. 1st ed. London: Springer London, 2011, pp. 177–208. ISBN: 9781849964494.

[24] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *"Deep Learning"*. http://www.deeplearningbook.org. MIT Press, 2016.

[25] G. Cybenko. "Approximation by superpositions of a sigmoidal function". eng. In: *Mathematics of Control, Signals and Systems* 2.4 (1989), pp. 303–314. ISSN: 0932-4194.

[26] Kurt Hornik. "Approximation Capabilities of Multilayer Feedforward Networks". In: *Neural Netw.* 4.2 (Mar. 1991), pp. 251–257. ISSN: 0893-6080. DOI: 10.1016/0893-6080(91)90009-T. URL: http://dx.doi.org/10.1016/0893-6080(91)90009-T.

[27] Machine Learning Cheatsheet. *"Backpropagation"*. URL: https://ml-cheatsheet.readthedocs.io/en/latest/backpropagation.html.

[28] Sebastian Raschka. *"Machine Learning FAQ"*. URL: https://sebastianraschka.com/faq/docs/closed-form-vs-gd.html.

[29] Tim Dettmers. *"Understanding Convolution in Deep Learning"*. URL: http://timdettmers.com/2015/03/26/convolution-deep-learning/.

[30] Rafael Padilla. *"Object Detection Metrics"*. URL: https://github.com/rafaelpadilla/Object-Detection-Metrics.

[31] Kasper Breistein. *"Subsea Object Detection and Classification Using Convolutional Neural Network. Project Thesis, Fall semester"*. 2018.

[32] Clas Ohlson. *"Hammer 12 oz"*. URL: https://www.clasohlson.com/no/Hammer-12-oz/40-9716.

[33] Raspberry Pi. *"Camera Module Hardware Specification"*. URL: https://www.raspberrypi.org/documentation/hardware/camera/.

[34] BlueRobotics. *"Raspberry Pi Camera Module v2 w/ Wide Angle Lens"*. URL: https://www.bluerobotics.com/store/sensors-sonars-cameras/cameras/cam-rpi-wide-r1/.

[35] BlueRobotics. *"BlueROV2"*. URL: https://www.bluerobotics.com/store/rov/bluerov2/bluerov2/.

[36] Massachusetts Institute of Technology. *"LabelImg Object Detection Tool"*. URL: https://github.com/tzutalin/labelImg.

[37] Google Inc. *"Google Colaboratory"*. URL: https://colab.research.google.com/.

[38] Janet Swift. *"Google Opens Doors To Its Colaboratory"*. URL: https://www.i-programmer.info/news/105-artificial-intelligence/11335-google-opens-doors-to-its-colaboratory.html.

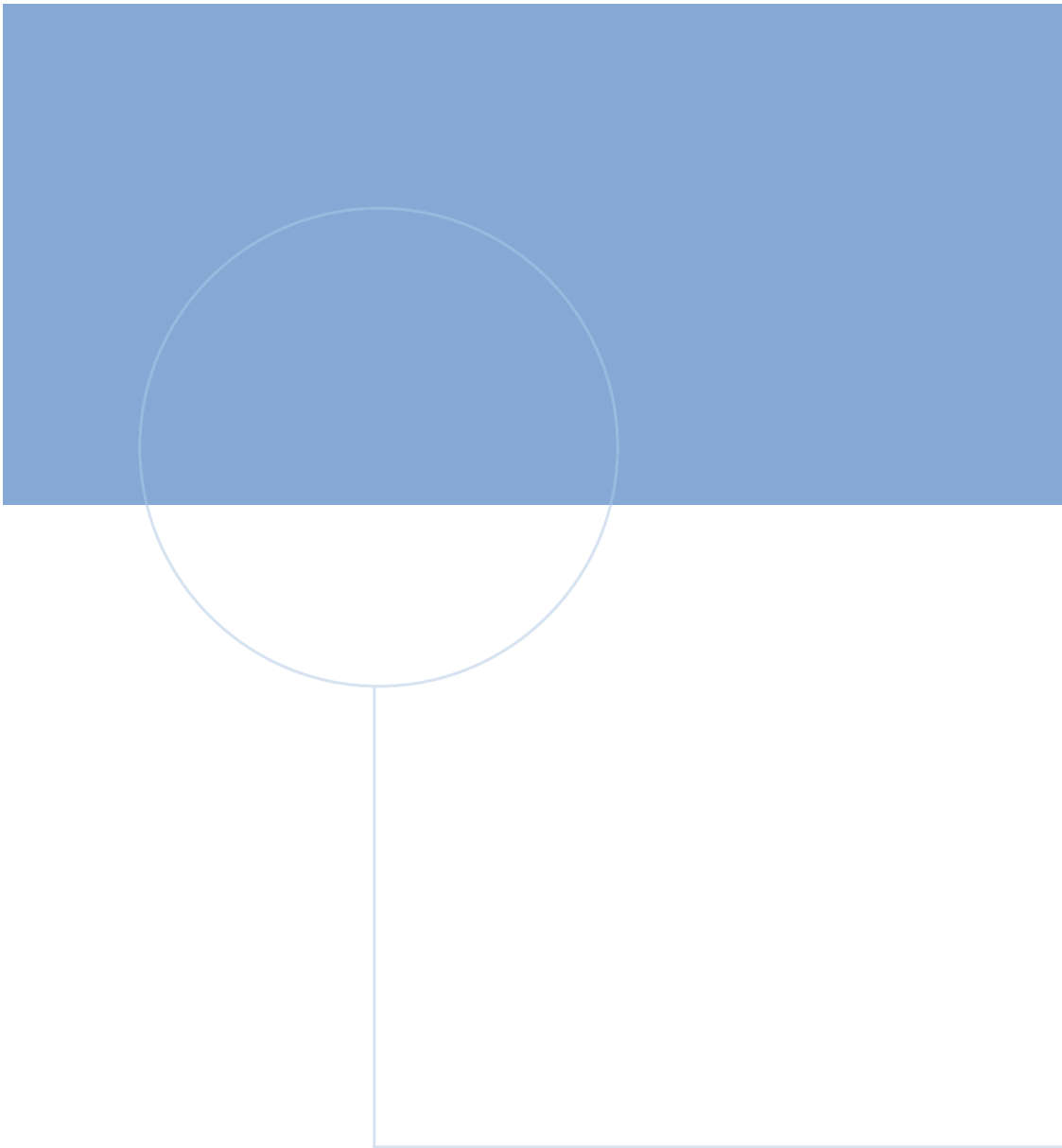
[39] Ryan Smith. *"NVIDIA Launches Tesla K80, GK210 GPU"*. URL: https://www.anandtech.com/show/8729/nvidia-launches-tesla-k80-gk210-gpu.

[40] Cade Metz. *"Google Just Open Sourced TensorFlow, Its Artificial Intelligence Engine"*. URL: https://www.wired.com/2015/11/google-open-sources-its-artificial-intelligence-engine/.

[41] Trieu H. Trinh. *"Darkflow Repository"*. URL: https://github.com/thtrieu/darkflow.

[42] Tsung-Yi Lin et al. "Microsoft COCO: Common Objects in Context". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2014), pp. 3686–3693. ISSN: 10636919. DOI: 10.1109/CVPR.2014.471. arXiv: 1405.0312v3.

[43] Rafael Padilla. *"Metrics for object detection"*. URL: https://github.com/rafaelpadilla/Object-Detection-Metrics.



NTNU
Norwegian University of
Science and Technology