Automated Detection and Classification of Red Roman in Unconstrained Underwater Environments Using Mask R-CNN

Christopher Conrady

Supervisor: Dr Şebnem Er

Co-Supervisor: Associate Professor Colin G. Attwood

Submitted in partial fulfilment of the requirements for the degree of
Master of Science in Data Science
in the
Department of Statistical Sciences
Faculty of Science, University of Cape Town

July 14, 2021

Abstract

The availability of relatively cheap, high-resolution digital cameras has led to an exponential increase in the capture of natural environments and their inhabitants. Video-based surveys are particularly useful in the underwater domain where observation by humans can be expensive, dangerous, inaccessible, or destructive to the natural environment. Moreover, video-based surveys offer an unedited record of biodiversity at a given point in time – one that is not reliant on human recall or susceptible to observer bias. In addition, secondary data that is useful in scientific study (date, time, location, etc.) are by default stored in almost all digital formats as metadata. When analysed effectively, this growing body of digital data offers the opportunity for robust and independently reproducible scientific study of marine biodiversity (and how this might change over time, for example). However, the manual review of image and video data by humans is slow, expensive, and not scalable. A large majority of marine data has never gone through analysis by human experts. This necessitates computer-based (or automated) methods of analysis that can be deployed at a fraction of the time and cost, at a comparable accuracy. Mask R-CNN, a deep learning object recognition framework, has outperformed all previous state-of-the-art results on competitive benchmarking tasks. Despite this success, Mask R-CNN and other state-of-the-art object recognition techniques have not been widely applied in the underwater domain, and not at all within the context of South Africa. To address this gap in the literature, this thesis contributes (*i*) a novel image dataset of red roman (*Chrysoblephus laticeps*), a fish species endemic to Southern Africa, and (*ii*) a Mask R-CNN framework for the automated localisation, classification, counting, and tracking of red roman in unconstrained underwater environments. The model, trained on an 80:10:10 split, accurately detected and classified red roman on the training dataset ($mAP_{50} = 80.29\%$), validation dataset ($mAP_{50} = 80.35\%$), as well as on previously unseen footage (test dataset) ($mAP_{50} = 81.45\%$). The fact that the model performs equally well on unseen footage

suggests that it is capable of generalising to new streams of data not used in this research – this is critical for the utility of any statistical model outside of "laboratory conditions". This research serves as a proof-of-concept that machine learning based methods of video analysis of marine data can replace or at least supplement human analysis.

Acknowledgments

To my parents, to my family and friends, and to my supervisor, Dr Şebnem Er and co-supervisor, Assoc. Prof. Colin G. Attwood, you have given me the only thing in this life that really matters: time.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context

The observation of animals in their natural habitat is central to the study of animal ecology and remains the simplest method of gathering data on the abundance, distribution and behaviour of species. Finding, counting, and identifying animals, especially as they occur naturally in the world, is a key challenge. Most ecological studies are limited by the time and the cost of fieldwork by human observers, and this often limits the breadth and scope of ecological research (Weinstein, 2018*a*). Furthermore, active observation by humans is not preferable in many cases, where such observation is disruptive to wildlife or potentially dangerous to the observer (Weinstein, 2018*a*). In addition, animal presence is often sporadic, and depends on important but infrequently observed events, such as breeding, predation or mortality (Weinstein, 2018*a*). Under these conditions, active or direct observation by field observers is inefficient. New technologies are emerging – the use of trap cameras (activated by motion), satellite imagery, unmanned aerial vehicles ("drones"), inexpensive consumer-grade camera technology (action cameras, smartphones), etc. – are being used to supplement or replace existing data collection methods (Pimm et al., 2015).

Active observation by humans in marine environments is especially difficult. For example, most surveys of coral reef fish are based on underwater visual censuses (UVC) carried out by scuba divers (Villon et al., 2018). The accuracy of these manual visual surveys, however, are highly dependent on the experience of the divers and on dive conditions. A dive may not be feasible in extreme weather conditions or when water visibility is poor, for example, or at a depth that is not safe for all but the most

experienced divers. There are also physical and financial limitations to the duration of a dive. In addition, the identification of certain marine species by divers can be prone to biases (Villon et al., 2018), or require specialised training. Given these limitations, it is important to note that UVC and other marine data collection methods (controlled angling surveys (CAS), tagging, netting, trawling) can be replaced or supplemented with camera data (Villon et al., 2018).

Video-based surveys provide estimations of fish abundance and species diversity similar to CAS and UVC-based surveys (Dando, 2020). Video-based methods can be used to overcome the limitations of human-based surveys (depth, dive duration, diver bias) (Dando, 2020). They also provide a permanent record for which analysis can be independently reproduced, which forms the basis of sound scientific study ((Ditria et al., 2020), and (Villon et al., 2018)).

The use of baited remote underwater video (BRUV) is increasingly being used as a versatile, cost-effective sampling apparatus to supplement or replace existing marine sampling methods like UVC, CAS and others (Dando, 2020). A BRUV should at least be equipped with some type of bait delivery system, one or more mounted cameras, and a mechanism to return the device to the surface. The bait works to attract fish into the field of view of the camera(s) that are monitored from the surface remotely, or if this is not possible, returned to the surface to be reviewed. In this way, a BRUV records marine diversity, abundance and behaviour with minimal impact on surrounding flora and fauna. Unlike some other marine sampling techniques, BRUV does not require the removal of the marine organism being studied. It is a non-contact sampling method that does not cause physical harm to the study organisms. BRUV is thus one of the least impactful survey techniques available to marine scientists (Dando, 2020).

To reduce cost, labour and logistics of data collection, ecologists are increasingly turning to greater automation to locate, count and identify organisms in natural environments (Pimm et al., 2015). Much like the collection of data by human observers, the manual analysis of image and video data is often labor-intensive, time-consuming and extremely costly (Weinstein, 2018*b*). For scientific study to be scalable, the rate of data analysis must be at least equal to the rate of data acquisition. Two of the largest collections of marine data, housed under The Integrated Marine Observing System (IMOS) and National Oceanic and Atmospheric Administration (NOAA), report less than 5% and less than 2% of digital marine data, respectively, will ever go through expert analysis (Moniruzzaman et al., 2017). It is evident that an automated method

of analysis offering a comparable accuracy at a fraction of the time and cost is critical and should be considered a research priority.

1.2 Contributions of Research

The contributions of this research are:

- a novel image dataset of red roman (*Chrysoblephus laticeps*), a fish species endemic to Southern Africa.
- a Mask R-CNN framework that achieves accurate localisation, classification, counting and tracking of red roman in unconstrained underwater environments.

The red roman dataset will contribute towards building a larger community of free-to-use and publicly available marine datasets in South Africa, for which there is currently a distinct lack.

The Mask R-CNN framework will serve as a proof-of-concept that machine learning based methods of video analysis of marine data can replace or at least supplement human efforts that are slower, and more expensive. These automated techniques offer an opportunity for cheap, fast, robust and independently reproducible scientific study of marine biodiversity.

1.3 Overview

This dissertation comprises seven chapters, including the current chapter which serves to introduce and provide motivation for the work. Chapter 2 is a review of the literature related to the objectives of this thesis. Chapter 3 describes the novel dataset created for this thesis. Chapter 4 presents the methodology followed, while Chapter 5 details the implementation of these methods. Chapter 6 presents and discusses the results achieved. Finally, Chapter 7 concludes with the contributions of this thesis, limitations and further work.

Chapter 2

Literature Review

2.1 Object Recognition of Fish in Underwater Videos

Still used today, the most widely used method of analysis of marine ecological data is manual review by trained human experts. The majority of analysis of this data is concerned with extracting information on the abundance, distribution and behaviour of the studied marine organism(s). This task can be broken down into four sub-tasks, frequently performed on a target species: (*i*) locate, (*ii*) identify, (*iii*) count, and (*iv*) track. Any automated computer-based approach should be able to perform at least one of these requisite sub-tasks.

It is useful to make a distinction between two related problems within object recognition[1]: image classification, and object detection. *Image classification* is concerned only with the identification of a *single* marine organism within an image – sub-task (*ii*) only. *Object detection*, however, refers to both the localisation and classification of one or more objects in an image. Object detection methods can locate (*i*), identify (*ii*) and count (*iii*) marine organisms in an image. This concept can easily be extended to video data – in this case, the task of object detection is repeated for every frame in a video sequence. In this way, object detection allows for the tracking (*iv*) of a target marine organism as well. Since object detection can solve all four sub-tasks simultaneously – much like a human expert – it is typically more useful than image classification for the automated analysis of marine video data.

[1]In this thesis, the term object recognition will be used broadly to refer to the suite of computer vision tasks including image classification (class prediction of a single object in an image), object detection/ instance segmentation (localisation and class prediction of all objects in an image).

Conventional methods for object detection in computer vision (CV) relied on hand-engineered, low-level feature extraction from images. Scale-Invariant Feature Transform (SIFT) (Lowe, 1999), Haar Cascade (Viola & Jones, 2001), and Histogram of Oriented Gradients (HOG) (Dalal & Triggs, 2005) are favoured in literature for their performance in object detection problems. SIFT, Haar Cascade and HOG are feature detection algorithms – they are meant to extract a set number of desired features from a target image, but they do not themselves perform classification. Instead, the generated features are passed as inputs into a classifier algorithm, usually a Support Vector Machine (SVM), or Neural Network (NN). It is common in literature to refer to an object detection model as a concatenation of these two parts, for example, HOG-Linear SVM. This two-part process for object detection and identification is more recently being replaced by so called "end-to-end" object detection, in which both the feature extraction part and classification part are performed by a single algorithm.

Instead of hand-crafted features, Hinton et al. (2006) proposed using a deeper CNN architecture that learns features automatically using Deep Neural Networks (DNNs). Qin et al. (2016), Sun et al. (2018), and Wang et al. (2019) apply deep learning for image classification of fish in unconstrained underwater environments. All models are evaluated against the Fish Recognition Ground-Truth dataset (Boom et al., 2012) – a popular fish benchmarking dataset – and are therefore comparable. They demonstrate that CNN-based networks can consistently achieve a classification accuracy of over 90% across the 23 fish species evaluated. Furthermore, they demonstrate CNNs when tested against the same dataset outperform previous state-of-the-art two-stage classifiers that rely on hand-crafted feature extraction.

While the work of Qin et al. (2016), and Wang et al. (2019) is limited to image classification, Sun et al. (2018) proposed an extended framework that incorporates Gaussian Mixture Modelling (GMM) for object motion detection prior to classification. Candidate regions that may contain objects are identified and fed as inputs into the trained CNN for classification. In this way, a system for object detection of fish in underwater videos was developed.

This hybrid approach for object recognition is used by Salman et al. (2020) for automatic fish detection in underwater videos. They evaluate their proposed model on benchmark datasets derived from the Fish4Knowledge project (Boom et al., 2012), achieving an accuracy of over 80%. Knausgård et al. (2020) proposed a similar hybrid (or two-step approach) for object recognition. Instead of conventional computer vision

techniques like GMM, they deploy a Single Shot Detector (SSD). In the first step, You Only Look Once (YOLO) is used for object localisation. In the second step, candidate regions are fed into a CNN for classification. They note transfer learning and data augmentation are used to overcome limited training samples. Their solution achieves a maximum accuracy of 87.74% on the test set.

Li et al. (2015) were among the first to apply deep learning methods to fish object detection in underwater video. They used a modified Fast Region-based Convolutional Neural Network (Fast R-CNN) on LifeCLEF Fish Task 2014 (sub-task 1) (ImageCLEF, 2014), a video-based fish object recognition challenge. To reduce class imbalance, the original dataset was modified by excluding fish species with fewer than 600 annotations to produce a modified dataset of 24277 images across 12 fish species. Their Fast R-CNN model achieved a mAP accuracy of 81.4% averaged across the 12 species evaluated, outperforming by a margin of 11.2% a baseline Deformable Parts Model (DPM) – the most successful and widely used object detection at the time, the authors note.

The concept of Fast R-CNN was extended further in Faster R-CNN (Ren et al., 2015) by introducing a Region Proposal Network (RPN). Faster R-CNN merges the RPN and Fast R-CNN into a single network. Mandal et al. (2018) refer to Faster R-CNN as an end-to-end deep learning-based architecture, offering a single simplified pipeline for object detection. They apply Faster R-CNN to a dataset of 4909 images (12365 annotations) of 50 fish species extracted from action camera footage of marine waters across southeast Queensland. With little modification and the use of transfer learning, they note that Faster R-CNN achieves comparable accuracy – mAP_{50} of 82.4% – to previous state-of-the-art Fast R-CNN networks, but with reduced training and inference times, making Faster R-CNN more suitable for both post and real-time video analysis.

Mask R-CNN (He et al., 2017) extends on Faster R-CNN by adding a branch for predicting a segmentation mask on a target object, in parallel with the existing branch for predicting a bounding box. Mask R-CNN has achieved the highest performance output for deep learning object detection models (He et al., 2017). Ditria et al. (2020) demonstrate that the state-of-the-art performance of Mask R-CNN is transferable to marine ecology. The contributions of Ditria et al. (2020) are two-fold: (1) They demonstrate the use of Mask R-CNN on fish object detection in unconstrained underwater videos, and (2) They compare the performance of Mask R-CNN against a sample of human experts. They create a dataset of 6080 mask annotations limited to a single

ecologically important fish, luderick (*Girella tricuspidata*), from video footage collected on action cameras deployed in seagrass meadows in Queensland, Australia. The Mask R-CNN model, pre-trained on ImageNet, achieved a mAP_{50} accuracy of 92.5% on the test set, and a mAP_{50} accuracy of 93.4% on novel footage collected from an entirely different location. Furthermore, in a separate experiment, the authors demonstrate that the performance of their model is better and more consistent (lower variance) than trained human experts.

The results of Ditria et al. (2020) demonstrate that deep learning, and Mask R-CNN in particular, can be a more accurate tool than humans at estimating the relative abundance of fish species from underwater video footage, and that results are consistent and transferable across survey locations. This is critical for any automated computer-based model to be considered as a viable and practical alternative to human analysis of marine ecological data.

Given the success of deep learning object detection frameworks, this research proposes the use of the current state-of-the-art Mask R-CNN for fish object detection. The model will be trained on the red roman dataset introduced in the next chapter.

Chapter 3

Data Description

Submerged action cameras were used to collect video footage of red roman (*Chrysoblephus laticeps*) in the coastal waters of the Western Cape, South Africa. The video footage was manually trimmed to include only those video snippets that contained the target species. From this subset of video data, an image dataset was created by extracting still video frames at regular intervals. This chapter introduces the red roman image dataset created for use in this thesis.

For context, Section 3.1 presents a review of the most popular datasets used today for image classification and object detection problems. These standardised datasets are used to benchmark novel computer vision algorithms, or to improve on existing ones. The large majority of these datasets cater for common objects – a variety of everyday scenes that contain objects that belong to simple, broad categories (bird, car, aeroplane, etc.). By contrast, there are fewer datasets that cater for underwater marine objects. Section 3.2 introduces the contribution of this thesis, the red roman image dataset for fish detection in unconstrained underwater environments.

3.1 Benchmark Datasets and Standardised Challenges

Caltech 101 (Fei-Fei et al., 2004) was among the first standardised datasets for multi-category common object image classification. Caltech 101 – as the name suggests – contains 101 object classes, and typically 15-30 training images per class. Caltech 256 (Griffin et al., 2007) and TinyImages (Torralba et al., 2008) followed, offering incremental improvements to both the number of training images and the breadth of classes. While hundreds of standardised benchmark datasets have been released since

Caltech 101 in 2004, ImageNet (Deng et al., 2009), PASCAL VOC (Everingham et al., 2010), and COCO (Lin et al., 2014) stand out as the de facto benchmarking dataset choice for large-scale object recognition.

The PASCAL VOC Challenge (Everingham & Winn, 2011), and The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) (Russakovsky et al., 2015), have set the precedent for the standardised evaluation of common object recognition algorithms in the form of yearly competitions. Both PASCAL VOC and ILSVRC consists of two components: (1) a publicly available dataset, and (2) an annual competition. The dataset allows for the rapid development and comparison of object recognition algorithms by decoupling data collection from research efforts and allowing for a more focused approach to algorithm development, while the competitions and subsequent workshops provide a way to track, evaluate and recognise progress year on year.

While PASCAL VOC and ILSVRC stand out as the most popular challenges for common objects, there are several datasets (with standardised competitions built around them) that seek to expand on the success of PASCAL VOC and ILSVRC but with a narrower focus on a particular field of study. Labeled Faces in the Wild (Learned-Miller et al., 2016) for unconstrained face recognition, and KITTI (Geiger et al., 2013) for computer vision in autonomous driving, are two noteworthy examples. These datasets along with PASCAL VOC and ILSVRC are critical to drive progress in different areas of computer vision.

By comparison, there are only a handful of publicly available, good quality datasets that exist for underwater marine objects. This is in part due to the inherent difficulties of acquiring new high-resolution footage of underwater environments, and the lack of existing footage freely available on the internet (Wang et al., 2020). However, the availability of such datasets is critical to advance the progress of computer vision in underwater marine environments, much in the same way as ImageNet, PASCAL VOC, and COCO have done for common object image recognition.

3.2 Red Roman Dataset

3.2.1 Dataset Preparation

A set of GoPro action cameras mounted on a baited rig (BRUV) was used to collect video footage of red roman (*Chrysoblephus laticeps*)[1]. The camera was set to record at 720p or 1080p resolution at 30 frames per second. Video samples were taken at Betty's Bay (34.3475° S, 18.9205° E) and further up the South Coast at Still Bay[2] (34.3642° S, 21.4336° E). Ten random sites across Betty's Bay and Still Bay were sampled – sampling from multiple sites ensures the capture of a variety of backgrounds, natural lighting, water visibility, and fish angles.

The captured video was trimmed to include only footage of the target species. From this subset of video footage, an image dataset was created by extracting still video frames at regular intervals (generally, five frames for every one second of video). A further subset of images was hand-picked as the most suitable candidates for training; the selection criteria used is described below. Figure 3.1 is a representative sample of the type of images selected. The red roman in these images differ by size, horizontal/vertical translation, rotation, shape deformation, truncation, occlusion, blur, colour, noise, brightness, resolution (pixels-per-inch) and age[3]. The fact that animals are deformable objects, and occur randomly and sporadically in the wild makes the task of accurate object recognition much more difficult, and is a common problem in developing state-of-the-art object detectors for animals (Chen et al., 2014). One way to overcome these limitations is to include as much variation in the training set as possible. Moreover, red roman can face away from the camera (Figure 3.1, [*bottom row*]), side-on (Figure 3.1, [*middle row*]) or head-on (Figure 3.1, [*top row*]). This further increases the difficulty of accurate object recognition.

3.2.2 Dataset Details

The red roman dataset comprises 2015 images and 2541 annotations, and is available online on the GitHub repository for this thesis (*Finding Nemo*, 2020) (see Table 3.1 for

[1]The author is indebted to Colin Attwood (co-author) and his team for the collection of this footage, and for granting permission for its use. The contribution of this thesis is not the collection, but the processing, labelling and analysis of this raw footage.

[2]Also called "Stillbaai".

[3]Figure 3.1, (h) is a juvenile red roman and differs slightly in form from mature adults.

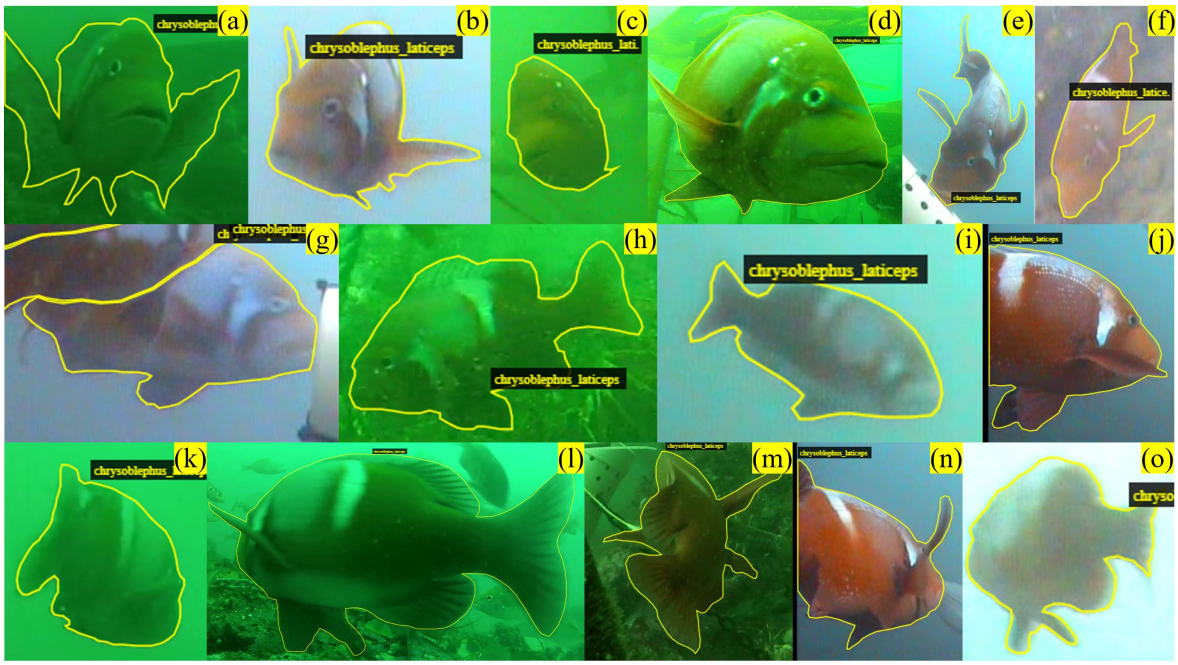

Figure 3.1: Visualisation of a representative sample of the red roman dataset. The red roman in these images differ by size (pixels per inch), horizontal/vertical translation, rotation, shape deformation, occlusion, blur, colour, noise, brightness, age, etc., and may be oriented to face the camera head-on *(top row)*, side-on *(middle row)*, or facing away from the camera *(bottom row)*.

a more detailed breakdown of the dataset). The annotations take the form of pixel-wise instance segmentation masks, from which a minimum bounding box can easily be computed (see Figure 3.2). In this way, the dataset is compatible with image classification, object detection as well as instance segmentation models. Annotations are stored in COCO JSON format (Lin et al., 2014), an increasingly popular and standardised format compatible with most machine learning frameworks including TensorFlow (Abadi et al., 2016) and PyTorch (Paszke et al., 2019).

Table 3.1: Breakdown of image and annotation count per site for the red roman dataset.

Site Name	Images	Annotations
Bettys Bay Site 10	179 (8.9%)	224 (8.8%)
Bettys Bay Site 28	151 (7.5%)	151 (5.9%)
Bettys Bay Site 56	373 (18.5%)	397 (15.6%)
Bettys Bay Site 59	514 (25.5%)	593 (23.3%)
Bettys Bay Site 68	176 (8.7%)	189 (7.4%)
Bettys Bay Site 83	126 (6.3%)	152 (6%)
Bettys Bay Site 87	221 (11%)	258 (10.2%)
Still Bay Site 1	96 (4.8%)	130 (5.1%)
Still Bay Site 2	74 (3.7%)	246 (9.7%)
Still Bay Site 3	105 (5.2%)	201 (7.9%)
	2015 (100%)	**2541 (100%)**

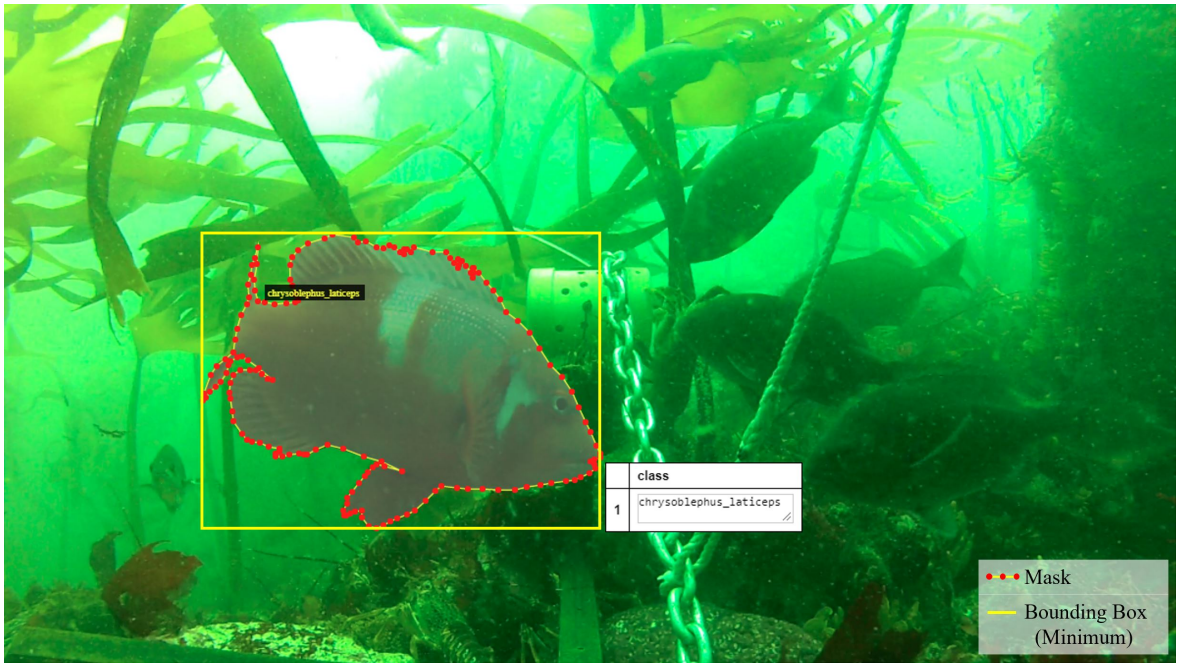

Figure 3.2: Illustration of a pixel-wise instance segmentation mask marking a single red roman from which a minimum bounding box is easily computed.

Chapter 4

Methodology

The next two chapters provide a detailed description of the method, split into the theoretical part (methodology) and the practical part (implementation). In this chapter, the key concepts of the algorithms used in this thesis are explained. This is followed in the next chapter with the implementation of these algorithms on the red roman dataset introduced in the previous chapter.

In the previous chapter, the first contribution of the thesis was presented: a novel image dataset of red roman that can be used to train and benchmark computer vision algorithms (like Mask R-CNN) in a South African setting. The second key contribution of this thesis is to present a framework that achieves accurate localisation, classification, counting and tracking of red roman in unconstrained underwater video footage. For the localisation and classification of red roman in video footage, Convolutional Neural Networks (CNNs) and Mask R-CNN are presented in Sections 4.1 and 4.2, respectively. For the counting and tracking of red roman in video footage, the Centroid Tracking algorithm is presented in Section 4.3. Together these algorithms comprise the framework presented in this thesis.

4.1 Convolutional Neural Network

Convolutional Neural Network (CNN) is a type of neural network that uses convolution in place of general matrix multiplication in at least one of its layers (Goodfellow et al., 2016). These convolutional layers adapt the neural network into a specialised type of machine learning architecture that is better suited for data that has a known grid-like topology, such as the 2-D pixel representation of a flat image.

Architecture

A CNN typically consists of an input layer, an output layer, and multiple hidden layers[1]. Together these layers comprise a feature extractor part, followed by a classifier part (see Figure 4.1). The most common layers are Convolutional (CONV), Activation (ACT, or [2]ReLU/SOFTMAX), Pooling (POOL, or MAXPOOL), Fully-connected (FC), Batch Normalisation (BN) and Dropout (DO). Almost all convolutional neural network architectures will consist of one or more of these layers. These layers, and the functions they each serve will be described in the subsequent subsections.

The architecture of CNNs can be described using a table, an illustrated diagram (see Figure 4.1), or a simple text diagram.

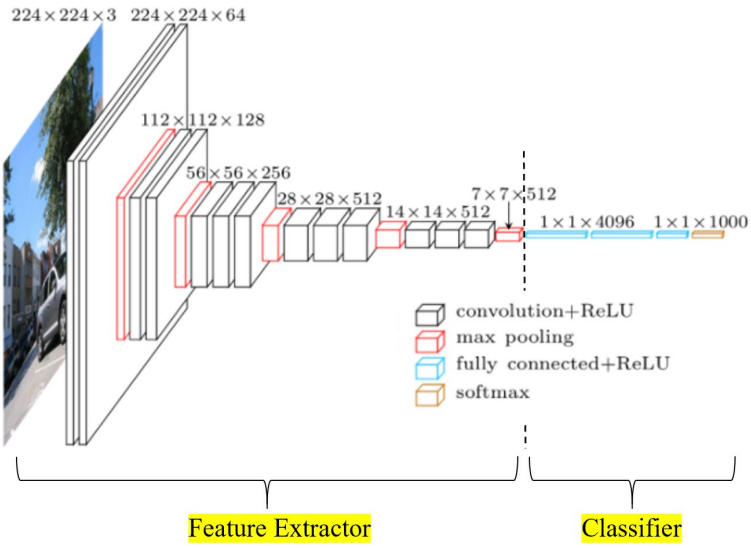

Figure 4.1: VGG16 architecture introduced by Simonyan & Zisserman (2014) in *Very Deep Convolutional Networks for Large Scale Image Recognition.* (Image credit: https://www.cs.toronto.edu/~frossard/post/vgg16/)

[1]Note that in the context of this thesis, a layer will refer to some discrete part or building block of the network architecture; nearly all machine learning libraries (including the popular Tensorflow library) follow this convention. An alternative is to consider a layer strictly as only those parts of the network for which a set of parameters are learned (convolutional and fully-connected layers, for example). Activation, pooling, batch normalisation and dropout layers do not contain parameters that are learned during the training process, but are included here as they are important in defining the architecture of CNNs and therefore often appear in network diagrams.

[2]The ReLU and SOFTMAX activation functions are so common that they can be referred to directly. Similarly, Max Pooling (MAXPOOL) is often written in place of the generic POOL.

For example, the VGG16 network architecture illustrated in Figure 4.1 can be described succinctly in text as,

$$\textbf{[}\boldsymbol{CONV} \Rightarrow \boldsymbol{MAXPOOL}\textbf{]} * \mathbf{2} \Rightarrow \textbf{[}\boldsymbol{CONV} \Rightarrow \boldsymbol{MAXPOOL}\textbf{]} * \mathbf{3} \Rightarrow \boldsymbol{FC} * \mathbf{3} \Rightarrow \boldsymbol{SOFTMAX}$$

(It is common to exclude batch normalisation (BN) and dropout (DO) for brevity.)

i. Convolution Layer (CONV)

In a broad sense, convolution refers to a mathematical operation performed on two functions of real-valued arguments that produces a third function that is some expression ("convoluted" form) of the original two functions. These convolution operations can in practice take on different forms depending on the field of study and the particular application. Within a deep learning context, and for the purpose of this thesis, a convolution (denoted by $*$) will be defined as (Goodfellow et al., 2016):

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n) K(m,n) \tag{4.1}$$

where I is a 2-D input image, and K a 2-D kernel (also referred to as a filter). The resulting output, S, is referred to as the feature map(s).

In practice, nearly all machine learning libraries implement the closely related cross-correlation function[3], defined as (Goodfellow et al., 2016):

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n) \tag{4.2}$$

where I, K, and S are similarly defined. A convolution implemented in this way can be considered simply as an element-wise multiplication of two matrices followed by a sum (see Figure 4.2).

Convolutions extend the traditional neural network and allow it to benefit from *sparse interactions* (or *local connectivity*) and *parameter sharing*. In a traditional neural network, each neuron (or unit) in the input layer (say, x_i) is connected to every output neuron in the subsequent layer (say, s_i) – this is called a fully-connected layer. There are two consequences of this setup that serve to demonstrate the concept of sparse interactions and parameter sharing. Each input neuron is said to interact with

[3]It is common in machine learning literature for the term convolution to be used in place of cross-correlation, even when the cross-correlation function is used, as will be the convention followed in this thesis.

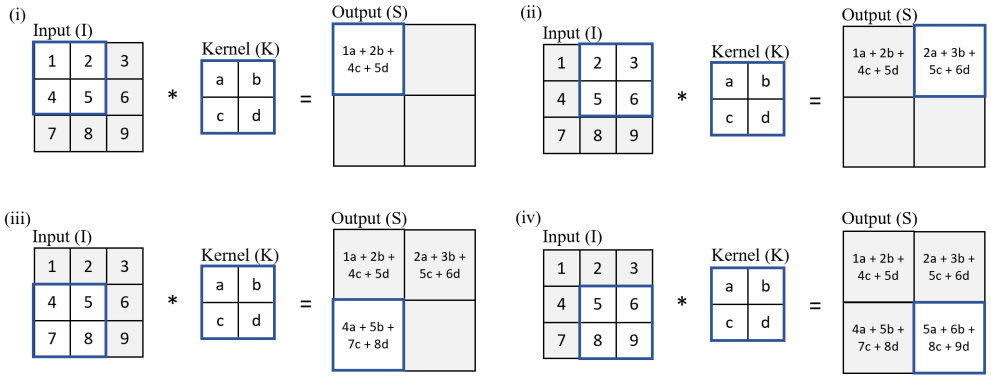

Figure 4.2: Step-by-step [(i) to (iv)] illustration of the element-wise multiplication of two square matrices of equal dimensions, I and K, to form the product matrix, S.

every output neuron, as shown in Figure 4.3 (*bottom row*). Since x_3 interacts with every s (*bottom left*), and s_3 interacts with every x (*bottom right*), this interaction is not sparse. Computationally, interaction is performed by matrix multiplication of a matrix of parameters (usually referred to as the weight matrix) with a separate parameter describing the interaction between each input neuron and each output neuron – and so parameters are not shared.

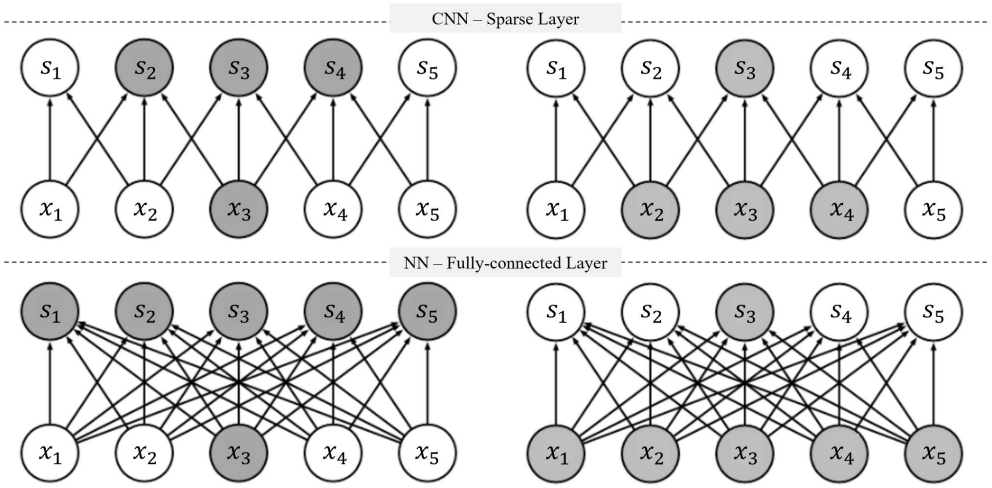

Figure 4.3: *(top)*: Each neuron in the input layer is connected to a small set of neurons in the output layer. *(bottom)*: Every neuron in the input layer is connected to every neuron in the output layer. (Image credit: Goodfellow et al. (2016).)

In a convolutional neural network, matrix multiplication of a fully-connected layer is replaced by convolution. In a convolution layer, interaction is limited by comparison,

achieved by setting the kernel to be smaller than the input, for example. In Figure 4.3, x_3 interacts with only three output neurons, s_2, s_3, and s_4 (*top left*); in turn, the output neuron s_3 is affected only by three input neurons, x_2, x_3 and x_4 (*top right*). In this way, convolution networks benefit from sparse interactions; furthermore, since each input neuron interacts with each output neuron through the same kernel, parameters are shared – a separate set of parameters are not learned for every location, as they are in a fully-connected neural network layer.

Since fewer parameters need to be stored and fewer operations need to be performed, sparse interaction and parameter sharing in a convolution layer is generally more efficient (and sometimes dramatically so) than dense matrix multiplication, significantly reducing memory requirements and improving statistical efficiency (Goodfellow et al., 2016).

ii. Activation Layer (ACT)

After every convolutional layer, the convolved image is passed through a non-linear activation layer. Generally, the output of the convolution layer is transformed by the activation layer (ACT) before it is passed into another convolution layer, or a fully-connected layer, etc. Since the activation function is applied in an element-wise manner, the output of the activation layer is the same dimension as the input.

The most commonly used non-linear activation functions are *sigmoid*, *tanh*, and *rectified linear unit (ReLU)*, given by Eq. 4.3, Eq. 4.4, Eq. 4.5, respectively.

$$Sigmoid : \theta(x) = \frac{1}{(1 + e^{-}x)} = \frac{e^x}{e^x + 1} \tag{4.3}$$

$$Tanh : \theta(x) = \frac{e^x - e^{-}x}{e^x + e^{-}x} \tag{4.4}$$

$$ReLU : \theta(x) = \begin{cases} x, & \text{for } x > 0 \\ 0, & \text{otherwise} \end{cases} \tag{4.5}$$

ReLU is a common choice in deep learning publications, perhaps because it is faster to compute and has been demonstrated to increase the speed of learning (Krizhevsky et al., 2012).

For classification, the output of a fully-connected layer at the end of the CNN network is almost always fed into a *softmax* activation function, as given by Eq. 4.6,

$$Softmax : \theta(\vec{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \tag{4.6}$$

where $\vec{x}$ is the input vector of weights, and K is the number of classes. The output of the softmax activation layer is a probability distribution vector ranging $[0, 1]$ over K predicted classes.

iii. Pooling Layer (POOL)

The pooling layer is effective at reducing the spatial dimensions of the input, while preserving important image information. The resultant output can have significantly less parameters, and in turn this reduces the overall computational complexity of the model. Furthermore, pooling can help control overfitting (Rosebrock, 2019*d*), and increase the robustness of the model to small translations of the input features (Goodfellow et al., 2016). Pooling layers are most commonly placed after consecutive convolution/activation layers. For example, in the VGG16 architecture (see Figure 4.1), pooling layers are placed after every ***CONV ⇒ ReLU*** sequence.

The pooling layer applies a pooling function to the output of the previous layer (usually a convolutional layer), using a "sliding window" called a filter that is dramatically smaller in dimension to the input layer. The pooling layer accepts two hyperparameters, receptive field size[4] and stride.

Max pooling is the most common pooling function, whereas average pooling and L2-norm pooling are less popular in literature. Max pooling is typically placed near the middle of a CNN architecture primarily to reduce spatial size, whereas average pooling is sometimes used in place of fully-connected layers (Rosebrock, 2019*d*). It is not always necessary to use pooling for spatial downsampling; Springenberg et al. (2014) demonstrate that using convolutional layers with large strides can replace pooling entirely and achieve adequate spatial downsampling.

iv. Fully-connected Layer (FC)

Fully-connected layers connect every neuron in one layer to every neuron in another layer. In a typical CNN architecture, FC layers are placed at the end of the network,

[4] Also called pool size, or filter size, or kernel size.

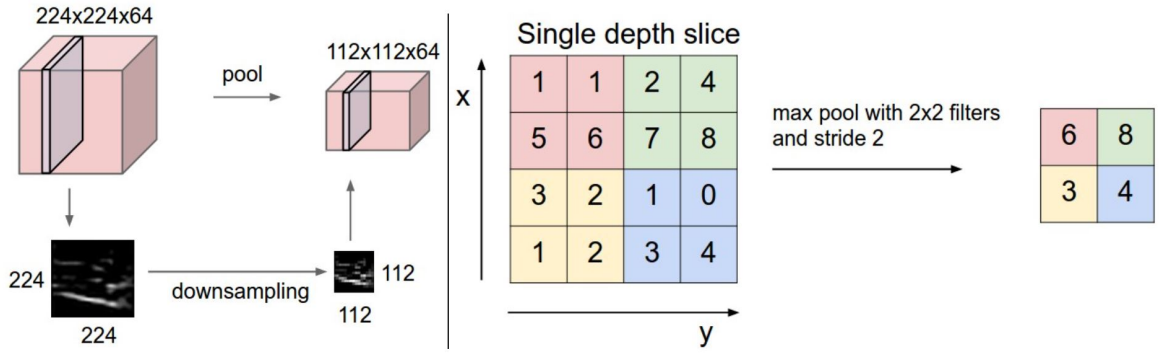

Figure 4.4: *(left)*: In this example, the input layer of size [224x224x64] is pooled with filter size 2 and stride 2 into an output layer of size [112x112x64]. *(right)*: The most common downsampling operation is max pooling, shown here with a stride of 2 and window/filter size of 2. (Image credit: https://cs231n.github.io/convolutional-networks/)

just before the final activation layer. For example, refer to the previously described VGG16 network,

$$[\boldsymbol{CONV} \Rightarrow \boldsymbol{MAXPOOL}] * 2 \Rightarrow [\boldsymbol{CONV} \Rightarrow \boldsymbol{MAXPOOL}] * 3 \Rightarrow \underline{\boldsymbol{FC} * 3} \Rightarrow \boldsymbol{SOFTMAX}$$

Since FC layers are almost always at the end of CNNs – directly following the convolution and pooling layers that are responsible for extracting feature maps – it can be said that every neuron in the FC layer *interacts* with the activations of all the previous layers in the network.

The output of the FC layers typically pass through a softmax activation function in order to compute the final output probabilities for each class – the classifier part of the CNN. The softmax activation function is given by Eq. 4.6.

v. Batch Normalisation Layer (BN)

Batch normalisation layers (BN) normalise (re-centre and re-scale) the activations of a given input layer before passing it into the next layer in the network. The resulting activation will have a mean of zero and a variance of one, and is said to be normalised. In practice, batch normalisation is thought to smooth the objective function (Santurkar et al., 2018), which can help to prevent overfitting, increase classification accuracy and reduce training time in some cases (Rosebrock, 2019*d*).

Since it is impractical to conduct batch normalisation over the entire training set, normalisation is restrained to mini-batches of the training set. Let B denote a mini-batch of size m of the whole training set. For a layer of the network with d-dimensional

input, $x = (x^{(1)}, ..., x^{(d)})$, each dimension of its input is then normalised separately as (Rosebrock, 2019*d*):

$$\hat{x}_i^{(K)} = \frac{x_i^{(K)} - \mu_B^{(K)}}{\sqrt{\sigma_B^{(K)^2} + \epsilon}}, \; where \, k \in [1, d] \, and \, i \in [1, m] \tag{4.7}$$

Eq. 4.8 and Eq. 4.9 describe the per-dimension *mean* and *variance*, respectively. Note that in practice ϵ is set to an arbitrarily small positive constant to avoid dividing by zero.

$$\mu_B = \frac{1}{m} \sum_{i=1}^{m} x_i \tag{4.8}$$

$$\sigma_B^2 = \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_B)^2 \tag{4.9}$$

vi. Dropout Layer (DO)

For each mini-batch in a training set, dropout layers randomly disconnect inputs from the preceding layer to the subsequent layer in the network architecture, temporarily preventing interaction between these neurons. After the forward and backward pass are computed for the mini-batch, the dropped connections are reconnected, and another set of connections to drop are sampled. The rate of random sampling is predefined for each dropout layer in the network.

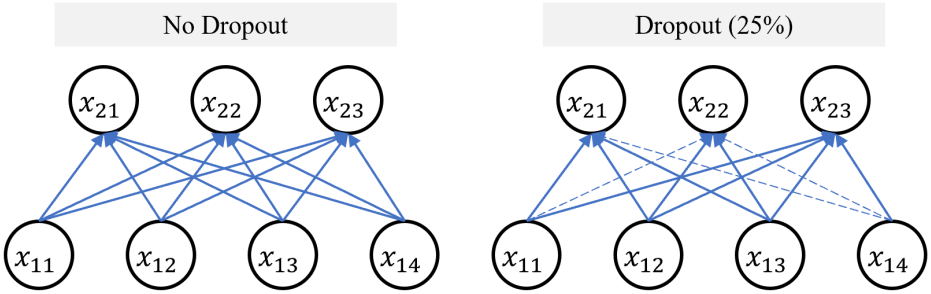

Figure 4.5: *(left)*: An FC layer with no dropout. There are 3 connections from each x_{1i} node to each x_{2i} node for a total of 12 connections. *(right)*: Connections are randomly disconnected with probability $p = 0.25$ between two FC layers for a given mini-batch. In this case, 3 connections (dashed lines) are randomly dropped.

Dropout layers are one of the simplest forms of regularisation, and can help prevent overfitting on the training data, increasing the ability of the model to generalise to unseen data (Rosebrock, 2019*d*). It is most common to place dropout layers between fully-connected layers, as this simple CNN text diagram illustrates:
CONV ⇒ ReLU ⇒ POOL ⇒ FC ⇒ DO ⇒ FC ⇒ DO ⇒ FC ⇒ SOFTMAX

4.2 Mask R-CNN

The Mask R-CNN algorithm was first introduced by He et al., in their 2017 paper by the same name, *Mask R-CNN* (He et al., 2017). The algorithm is an extension of state-of-the-art object detection algorithms published by Girshick et al., namely R-CNN (Girshick et al., 2014), Fast R-CNN (Girshick, 2015) and Faster R-CNN (Ren et al., 2015). In turn, these algorithms build on the image classification algorithms popularised by LeCun et al. (1998) in their seminal paper *Gradient-based learning applied to document recognition* (LeCun et al., 1998), known collectively as Convolution Neural Network (CNN, for short). In the previous section, the CNN algorithm was described in detail as it forms the backbone of Mask R-CNN. In this section, Mask R-CNN and related computer vision problems will be described.

Background

Figure 4.6 familiarises four common problems in computer vision (CV). The task of *image classification* is to correctly predict the category (or class) of an image. An image classification algorithm will typically predict a single label for an entire image. In Figure 4.6 (a), for example, the predicted class of the image is bottle (or cup, or cube) – the strongest prediction of the three is chosen.

Convolutional Neural Network (CNN) is a state-of-the-art deep learning algorithm for image classification problems. By default, CNNs are set up to predict a single class belonging to an image, and are not able to solve the problems in Figure 4.6 (b)-(d)[5]. By comparison, algorithms that are capable of *object localisation*, can predict multiple labels per image (although they may predict one or none) each associated with the

[5]This does not mean that CNNs cannot be used as object detectors. For example, an image could be divided into a set of smaller images that are systematically fed into a CNN for classification (see (Rosebrock, 2019*c*) for a worked example). Of course, this is simply a rudimentary implementation of Faster R-CNN.

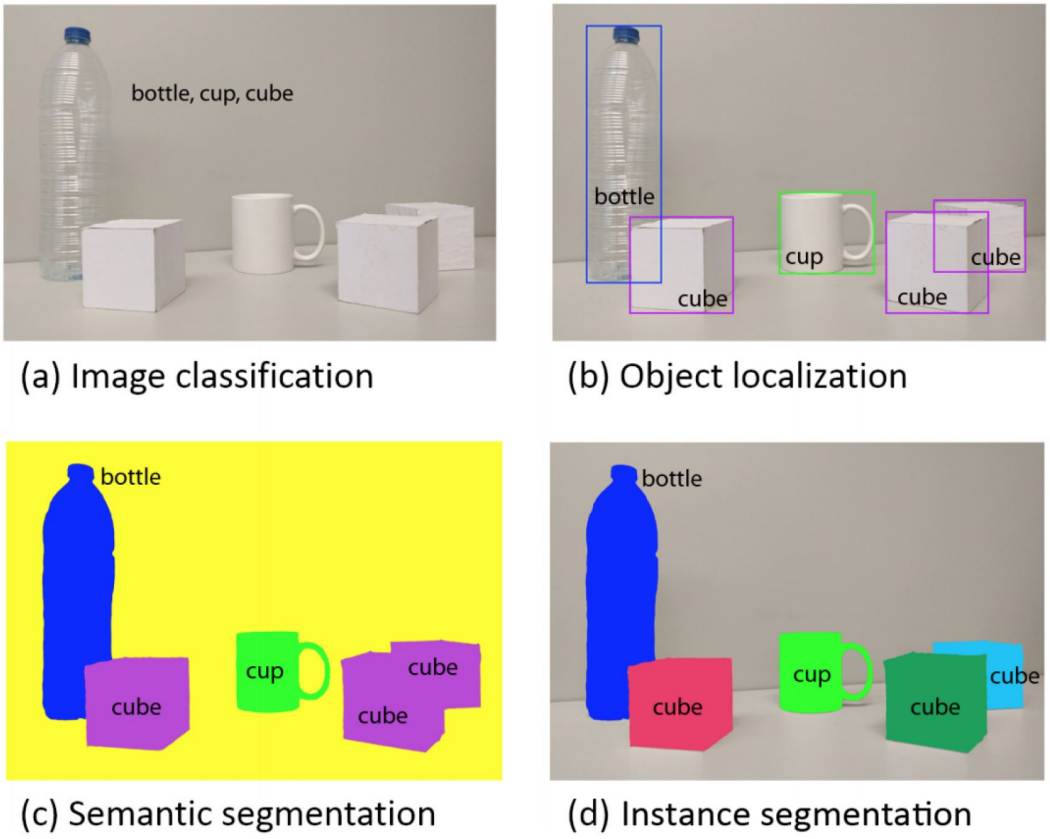

Figure 4.6: A visual representation of four common problem types in computer vision, namely, (a) Image Classification, (b) Object Localisation, (c) Semantic Segmentation, and (d) Instance Segmentation. (Image credit: Rosebrock (2019*a*).)

likelihood of an object detection. Mask R-CNN and R-CNN (Fast/ Faster) are all examples of object detection algorithms that are capable of object localisation.

The final two problem types are instance segmentation and semantic segmentation. The objective of *instance segmentation* is to predict a pixel-wise segmentation (or mask) for every unique object within an image, while *semantic segmentation* seeks to associate every pixel of an image – including the background – with a class label. Note that semantic segmentation does not differentiate between individual objects of the same class; it does not try to separate objects that may be lumped together, or partially occluded, but belong to the same class.

The Mask R-CNN architecture used in this thesis is an example of an instance segmentation algorithm.

Architecture

There is no standard Mask R-CNN architecture – the design of a Mask R-CNN network will be influenced by the type of problem that is to be solved. It is more useful then to describe the key design features that are common in most Mask R-CNN architectures. Some of these have already been discussed. For example, Mask R-CNN shares parts of its architecture with that of CNN – this includes activation, pooling and fully-connected layers, and may also include batch normalisation and dropout layers. In particular, convolution layers are critical for the Mask R-CNN to be able to classify objects. It is for this reason that any discussion of Mask R-CNN will include some reference to CNNs and the underlying convolution operation. In the same way, while there is a great deal of variation in Mask R-CNN architectures, three principal features stand out: RPN, RoIAlign and Mask Branch.

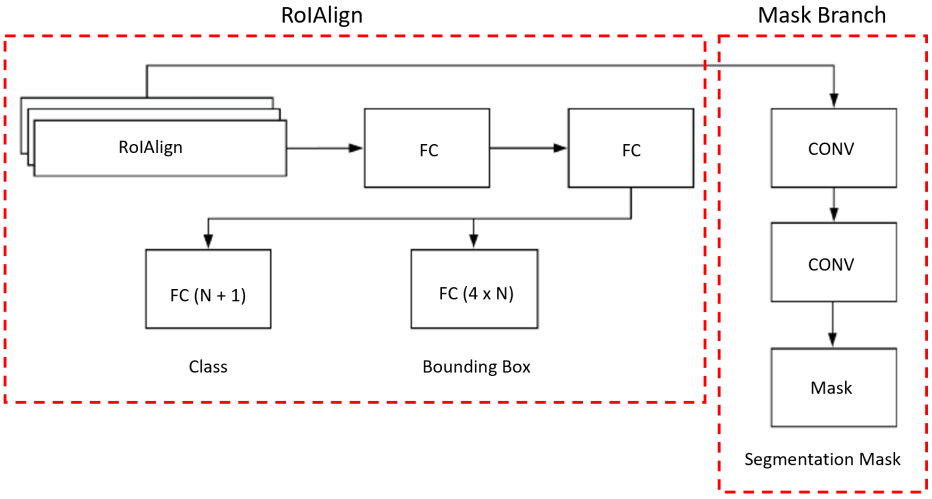

Figure 4.7: The output of the RoIAlign module is passed through two FC layers, then through two independent FC layers, *Class* and *Bounding Box*. The FC layer, *Class*, has a node for each of the class label predictions plus an additional label for the background $(N+1)$. The FC layer, *Bounding Box*, has four nodes per class, representing a four-value system for defining a rectangular bounding box. In parallel, the output of the RoIAlign module is passed through a Fully Convolution Network (FCN) which encodes K (number of object detections) binary masks of resolution $m \times m$. (Image credit: Rosebrock (2019*a*).)

i. RPN

Mask R-CNN is able to localise multiple objects within a larger image. To do this, Mask R-CNN leverages a Region Proposal Network (RPN) to generate regions of an image that potentially contain an object. In this way, multiple parts ("images within images") of a larger image are evaluated. It is common to refer to these candidate regions, or candidate bounding boxes, as regions of interest (or RoIs). Each RoI is ranked based on how likely it is that the given region could potentially contain an object of interest, and the top N RoIs by confidence are retained. The RPN is generally considered to be the first of a two-stage procedure.

In the second stage, the RoIAlign module extracts features from each candidate region before performing classification and bounding-box regression. In parallel, the output of the RoIAlign module is fed into an additional branch referred to loosely as the "mask branch" which outputs a binary mask for each RoI – pixels that belong to the object are encoded with the value one and zero otherwise. By design, the parallel branches ensure that classification does not depend on mask or bounding-box predictions (and vise versa). Three independent outputs for each candidate object are thus produced: a class label, a bounding-box offset, and an object mask.

ii. RoIAlign

RoIAlign is shown empirically to be an improvement over its predecessor, RoIPool, used in Fast R-CNN (Girshick, 2015), by reducing the misalignment between the RoI (input) and the extracted features (He et al., 2017). Since the region proposals are by some factor smaller than the input image, the proposals need to be scaled down to match the dimensions of the feature maps. In RoIPool, each x (width) and y (height) co-ordinate is divided by some factor to achieve downscaling of the input image. Since these co-ordinates (or anchor points) cannot be represented as continuous values they are truncated to discrete values. This process, called quantisation, is performed twice in RoIPool; first, in the mapping process (computing boundaries), and second in the pooling process (computing bins). The truncation of a continuous value to a discrete value introduces an inevitable loss in precision, and this creates a small but significant[6]

[6] He et al. (2017) notes that while this [misalignment] may not impact classification, which is generally robust to small translations of the input, it has a large negative effect on predicting pixel-accurate masks, necessitating RoIPool.

misalignment between the RoI and the extracted features.

By contrast, in RoIPool the exact values of the input feature at four regularly sampled locations in each RoI bin are computed using bilinear interpolation (see Figure 4.8), then aggregated (using max or average pooling). No quantisation is performed.

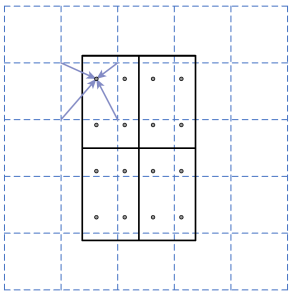

Figure 4.8: In RoIAlign, the value of each sampling point (black dots) within an RoI (black lines) is computed using bilinear interpolation from the nearby points on the blue dashed grid (feature map). No quantisation is performed. (Image credit: He et al. (2017).)

iii. Mask Branch

The output of RoIAlign is fed into what is referred to by He et al. (2017) as the "mask branch", which is responsible for predicting segmentation masks for target objects contained in an input image. A mask encodes an input object's spatial layout – pixels that belong to the object are encoded with the value one and zero otherwise. Since the values of the pixels of the original image are not retained, the image content is lost. Instead, the binary encoding of the object's spatial layout can be used to produce an outline or monochromatic fill of the object that is usually overlaid on the original image – this outline or monochromatic fill serves as the mask of the object.

In the Mask Branch, K masks, representing the number of object detections, of resolution $m \times m$ are predicted from each RoI using a Fully Convolutional Network (FCN). This is preferred to using a fully-connected layer (like for the class and bounding-box predictions) as it requires fewer parameters and is shown to be more accurate in experimentation (He et al., 2017). Moreover, this fully convolutional representation of the object mask is more faithful to the natural 2-D representation of an object within a flat image, rather than the 1-D vectorised format required by a fully-connected network that cannot be easily represented on a 2-D plane.

Training

During training, a multi-task loss function on each sampled RoI is defined as the sum of three separate task losses: the classification log loss ($\mathcal{L}_{cls}$), localisation log loss ($\mathcal{L}_{bbox}$) and segmentation mask log loss ($\mathcal{L}_{mask}$) (He et al., 2017):

$$\mathcal{L} = \mathcal{L}_{cls} + \mathcal{L}_{bbox} + \mathcal{L}_{mask} \tag{4.10}$$

The classification log loss is defined as (Girshick, 2015):

$$\mathcal{L}_{cls}(p, u) = -\log p_u, \text{ for the true class } u, \text{ against } p, \text{ where} \tag{4.11}$$

$$p = (p_0, ..., p_K), \tag{4.12}$$

is the discrete probability distribution (per RoI), over $K + 1$ categories, and

$$u = (u_0, ..., u_K), \tag{4.13}$$

is the set of ground-truths (per RoI) over $K + 1$ categories.

For the second task loss, each training RoI is labelled with a ground-truth class u and a ground-truth bounding-box regression target, for which the bounding-box task loss is defined as (Girshick, 2015):

$$\mathcal{L}_{bbox}(t^u, v) = \sum_{i \in \{x,y,w,h\}} smooth_{L_1}(t_i^u - v_i) \tag{4.14}$$

in which,

$$smooth_{L_1}(x) = \begin{cases} 0.5x^2, & \text{if } |x| < 1, \\ |x| - 0.5, & \text{otherwise} \end{cases} \tag{4.15}$$

and t_u is a tuple of predicted bounding-box offsets for class u,

$$t^u = (t_x^u, t_y^u, t_w^u, t_h^u), \tag{4.16}$$

and v is a tuple of ground-truth bounding-box offsets,

$$v = (v_x^u, v_y^u, v_w^u, v_h^u), \tag{4.17}$$

where x, y are the centre co-ordinates of the bounding box, and w and h are the width and height of the bounding box, respectively.

Lastly, the mask branch has a Km^2 dimensional output for each RoI, which encodes K binary masks of resolution $m \times m$, one for each of the K classes. To this a per-pixel sigmoid function is applied, and the task loss for the segmentation mask is simply the average binary cross-entropy loss, defined as (He et al., 2017):

$$\mathcal{L}_{mask} = -\frac{1}{m^2} \sum_{1 \leq i,j \leq m} [y_{ij} \log \hat{y}_{ij}^k + (1 - y_{ij}) \log(1 - \hat{y}_{ij}^k)] \tag{4.18}$$

where y_{ij} is the label of a cell (i, j) in the $m \times m$ region belonging to the true mask, and $\hat{y}_{ij}^k$ is the predicted label of the same cell in the mask learned for the ground-truth class, k. Note that since the objective function is a negative log loss, the objective is to minimise the loss. Since the log loss and task prediction probability are inversely proportional, minimising the log loss is analogous to maximising the predictive accuracy of the model.

Inference

The multi-task loss function described in Eq. 4.10 can be used to evaluate the performance of Mask R-CNN during training. It is also common to use log loss as a metric to evaluate the accuracy of Mask R-CNN when making predictions. There are two additional metrics for measuring inference performance of Mask R-CNN[7], namely Intersection over Union (IoU) and mean Average Precision (mAP).

Intersection over Union (IoU)

The Jaccard similarity coefficient, or Jaccard Index, is a statistical measure of similarity between two sample sets. The Jaccard index is described mathematically by Fletcher et al. (2018):

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}, \tag{4.19}$$

where A and B are finite sets.

[7]The metrics log loss, IoU and mAP can be used to evaluate the performance of any object detection algorithm that produces a bounding-box prediction and class label. This includes Mask R-CNN but also Fast/Faster R-CNN, YOLO, etc.

For two identical sets, $J(A, A) = 1$, and for two mutually exclusive sets, X and Y, $J(X, Y) = 0$. It follows that a score close to 1 describes two sets that are extremely similar, a score close to 0 describes two sets that are extremely dissimilar, while a score of 0.5 describes two sets that are as similar as they are dissimilar.

Intersection over Union (or IoU) is considered to be equivalent to the Jaccard index, since they both describe the ratio between two intersected and two unioned sample spaces. However, the term IoU is more common in object localisation literature. In object localisation, two sets of bounding boxes – or masks[8] – are defined: a ground-truth (or label) set, say Y, and a prediction set , say $\hat{Y}$. IoU describes the relationship between these two sets as,

$$IoU = \frac{Y_{area} \cap \hat{Y}_{area}}{Y_{area} \cup \hat{Y}_{area}}, \tag{4.20}$$

where Y_{area} is the area enclosed by the ground-truth bounding box and $\hat{Y}_{area}$ is the area enclosed by the predicted bounding box (see Figure 4.9 for an illustration of Eq. 4.20).

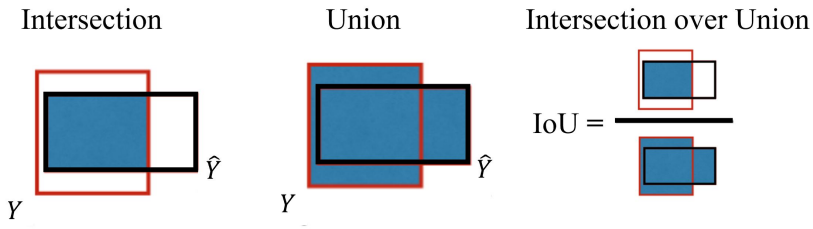

Figure 4.9: IoU is simply the area where a prediction and ground-truth box overlap with each other (intersection) divided by the total area covered by the prediction and ground-truth box (union). Image credit: https://www.oreilly.com/library/view/hands-on-convolutional-neural/

Similar to the Jaccard Index, an IoU value of 0 indicates that no part of the ground-truth and prediction area overlap – the weakest prediction accuracy possible. A value of 1 indicates a perfect overlap of the ground-truth and prediction area – the strongest prediction accuracy possible. Figure 4.10 shows how IoU rewards predicted bounding boxes for heavily overlapping with the ground-truth bounding box, while implicitly penalising those with less overlap.

[8]While IoU is often formally described using bounding boxes, it can easily be applied to segmentation masks (either through the minimum bounding box derived from the segmentation mask, or applied directly to the mask itself).

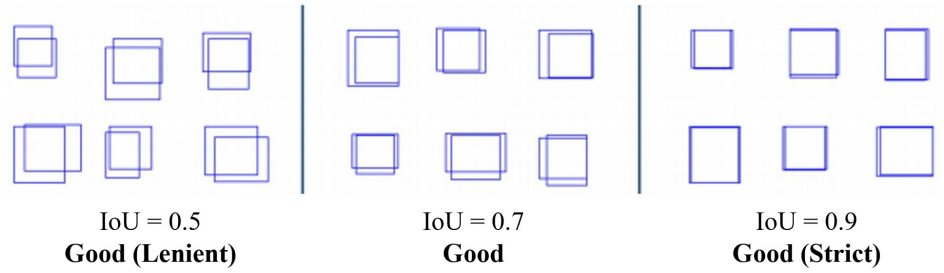

Figure 4.10: An illustration of random overlapping bounding-box sets, ranging in value from an IoU of 0.5 to an IoU of 0.9. An IoU of 0.5, while common in literature, is the minimum criteria for a good overlap, and is usually set as the threshold for a positive prediction. However, an IoU of 0.7 is a more reasonable compromise between very loose (IoU of 0.5) and very strict (IoU of 0.9) overlap values (Zitnick & Dollár, 2014). (Image credit: Zitnick & Dollár (2014).)

An IoU score greater than 0.5 (or 50%) means that more than half of the predicted box overlaps the ground-truth box. Generally, an IoU score greater than 0.5 is considered to be a good prediction, and is usually set as the threshold value (Rosebrock, 2019*b*).

A *threshold*, say p, allows IoU to be expressed on a discrete scale – usually as a *true* detection (contains an object), or a *false* detection (does not contain an object). Formally, this is given by Eq. 4.21.

$$class(IoU) = \begin{cases} True, & \text{if IoU} \geq p \\ False, & \text{if IoU} < p \end{cases} \tag{4.21}$$

mAP

mAP is defined by the relationship between Precision (P) and Recall (R). *Precision* is the proportion of correctly predicted objects from the positive class. A correctly predicted object is one that scores above the set IoU threshold. Precision is defined as (Rosebrock, 2019*b*):

$$Precision(P) = \frac{True\ Positive\ (TP)}{True\ Positive\ (TP) + False\ Positive\ (FP)} \tag{4.22}$$

For example, in this thesis a precision value of 0.5 indicates that when the model predicts an object as a red roman, it is correct 50% of the time.

Recall is the proportion of correctly predicted objects from *all* predicted objects, and is defined as (Rosebrock, 2019*b*):

$$Recall(R) = \frac{True\ Positive\ (TP)}{True\ Positive\ (TP) + False\ Negative\ (FN)} \tag{4.23}$$

For example, a recall value of 0.5 indicates that the model correctly identified 50% of all red roman.

The mean average precision value (mAP) quantifies the ability of the model to fit a segmentation mask (Everingham et al., 2010):

$$mAP = \int_0^1 P(R)dR \tag{4.24}$$

$P(R)$ is a measure of the precision of the model at a given level of recall. Eq. 4.24 describes mathematically the precision-recall curve. In practice, mAP, or the area under the precision-recall curve, is approximated using numerical methods, as is the convention in this thesis.

Finally, the notation mAP_{50}, or mAP_{70}, refers to mAP calculated with a threshold of 50% or 70%, respectively. In other words, a correctly predicted object (*True Positive* – TP) is defined here as one that scores an IoU above 50% or 70%, respectively.

4.3 Object Tracking (Centroid)

Given an initial set of objects, object tracking as a task involves:

i. assigning each unique object with an ID
ii. storing the position of the object(s) within a reference frame
iii. updating the position of the object(s), maintaining correct ID assignment

i. assigning each unique object with an ID

Suppose we are provided with an initial set of objects, as per Figure 4.11 (*left*). Since this is the first frame observed (f_n), each object is assigned a unique ID. Note that object IDs are never recycled. For example, if a new object is detected in a subsequent frame (f_{n+1}), the new object is assigned an ID that has not been previously assigned (see Figure 4.11 [*right*]).

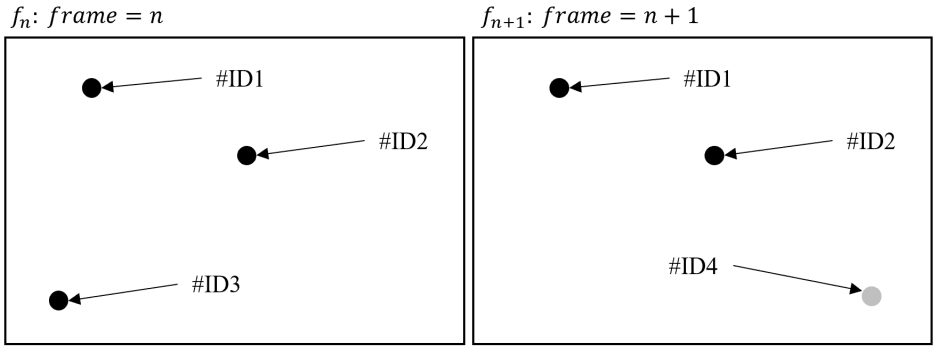

Figure 4.11: *(left)*: In the first frame, f_n, each object is assigned a unique ID. *(right)*: In the subsequent frame, f_{n+1}, a new object is detected and it is assigned an ID that has not been used before.

ii. storing the position of the objects within a reference frame

There are several ways to reference the position of an object within a frame, though the simplest and most common is to use an object's centroid. The centroid of an object is simply the geometric centre, and is calculated as the arithmetic mean of all points along the object's boundary. For objects of irregular shape, it is often easier to approximate the centroid by taking the geometric centre of a minimum bounding box (see Figure 4.12 [*left*]). Then, the centroid of the object is approximated as,

$$(c_1, c_2) = (\frac{w_{object}}{2}, \frac{h_{object}}{2}) \tag{4.25}$$

while the position of the object's centroid within a frame of reference is given by,

$$(p_1, p_2) = (d_1 + c_1, d_2 + c_2) \tag{4.26}$$

where (p_1, p_2) is the (x, y) co-ordinate pair on a Cartesian plane (see Figure 4.12 [*right*]).

iii. updating the position of the objects, maintaining correct ID assignment

In order for objects to be tracked, say from frame f_n to f_{n+1}, the objects in f_n need to be *linked* to (or associated with) the objects in f_{n+1}. Then, since the position of objects in frame f_n and f_{n+1} are known, the movement or relative change in the position of the objects can be easily computed. In this thesis, the criteria of minimum Euclidean distance is used to associate objects, and is illustrated by way of example (see Figure 4.13):

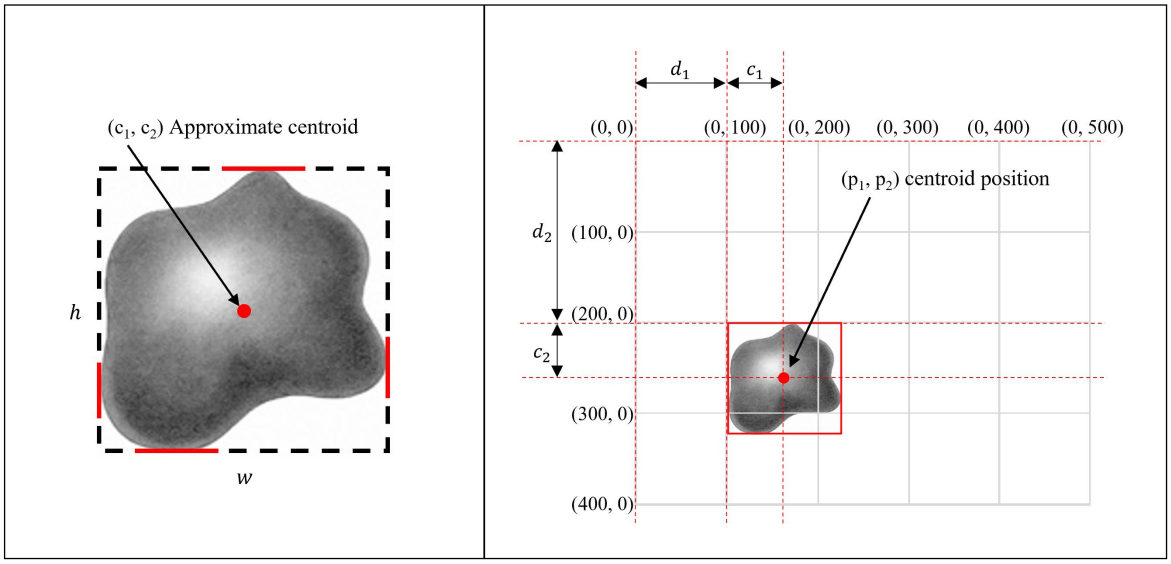

Figure 4.12: *(left)*: visual representation of the approximate centroid (c_1, c_2) of an irregular shape, by taking the geometric centre of a minimum bounding box. *(right)*: the relative position of an object within a frame of reference plotted on a Cartesian plane.

i. frame = n

In the initial frame, f_n, two objects are observed, represented by their respective centroids. Each object is assigned a unique ID, and the relative positions of the objects within the frame are stored.

ii. frame = n+1

In the subsequent frame, f_{n+1}, three objects are observed, and their positions stored. Before the movement of any object from frame f_n to f_{n+1} can be known, however, the objects in f_n need to be linked to the objects in f_{n+1}. First, the Euclidean distance is computed between objects in f_n and objects in f_{n+1}. (Note that calculating the distance between objects of the same frame is not necessary.) It is assumed that the two objects with the smallest Euclidean distance between frames f_n and f_{n+1} are identical. This is the primary assumption of the centroid tracking algorithm. That is, it is assumed that a given object will move between frames, but that the distance between this identical object will be smaller than its distance from all other objects.

Formally, let the position of objects in frame f_n be represented by vector $\vec{p}$, and the position of objects in frame f_{n+1} be represented by vector $\vec{q}$, then the Euclidean distance between any two points p_i and q_j can be calculated on a Cartesian co-ordinate plane by,

$$d_{ij}(p_i, q_j) = \sqrt{(p_{i1} - q_{j1})^2 + (p_{i2} - q_{j2})^2} \tag{4.27}$$

In Figure 4.13 (*ii*) three unidentified objects are marked by their respective centroids, labelled $\hat{A}$, $\hat{B}$ and $\hat{C}$. The Euclidean distances between $\hat{A}$, $\hat{B}$ and $\hat{C}$ (of frame f_{n+1}) and X and Y (of frame f_n) are computed. The result is that X_{f_n} is associated with $\hat{A}_{f_{n+1}}$, since the distance between these two centroids is shorter than the distances between all other pairs. Likewise, the object represented by centroid Y is associated with the object represented by centroid $\hat{B}$. Formally,

$$min(d_{11}, d_{12}, d_{13}) = d_{11}$$

$$\therefore X_{f_n} \equiv \hat{A}_{f_{n+1}}$$

and,

$$min(d_{21}, d_{22}, d_{23}) = d_{21}$$

$$\therefore Y_{f_n} \equiv \hat{B}_{f_{n+1}}$$

iii.—iv. frame = n+1 (cont...)

The final step is to assign new IDs to the remaining objects within the frame. Since $\hat{C}_{f_{n+1}}$ is not closest to any other object in f_n, it is considered to be a new object and is assigned a unique ID.

In an end-to-end object tracking framework, the process followed in steps (*i*)—(*iii*) is repeated for every selected frame in a video sequence, and in this way the movement of unique objects in a video stream are tracked.

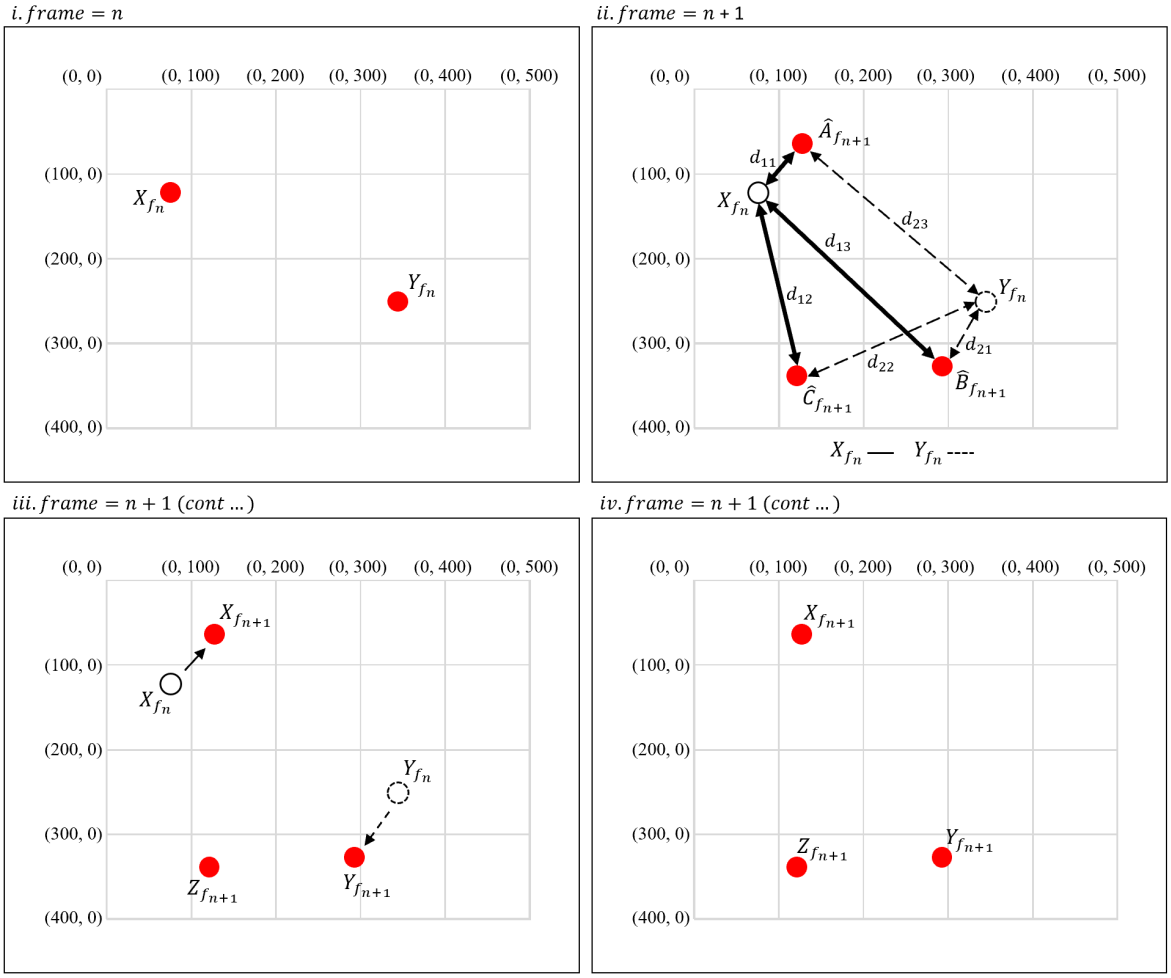

Figure 4.13: *(i. frame = n):* Each unique object is assigned an ID. *(ii. frame = n + 1):* The Euclidean distances between every "old" object in f_n and every "new" object in f_{n+1} are computed. *(iii. – iv. frame = n + 1(cont...)):* The objects in frame f_{n+1} are linked with the objects in frame f_n with the smallest Euclidean distance. The remaining object in f_{n+1} is assigned a new unique ID.

Chapter 5

Implementation

In the previous chapter, the theoretical concepts of the algorithms used in this thesis were covered. This chapter is dedicated to the practical implementation of these algorithms.

5.1 Overview

The layout of this chapter mimics the structure of the project. Figure 5.1 describes the high-level pipeline of the tasks involved. Sections 5.2.1, 5.2.2 and 5.2.3 discuss the preparation, labelling and format of the red roman image dataset, respectively. Section 5.3 describes the environment setup. Section 5.4 describes the model training and validation (which is an iterative process).

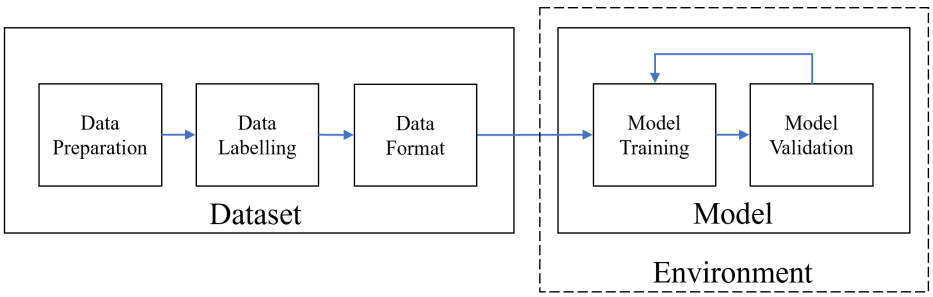

Figure 5.1: Project pipeline: Overview of the project workflow.

5.2 Red Roman Dataset

5.2.1 Data Preparation

Video footage of red roman (*Chrysoblephus laticeps*) in Betty's Bay and Still Bay in the Western Cape was collected. The footage was captured on GoPro action cameras in up to 1080p resolution at 30 frames per second. The captured video footage was trimmed using widely available video editing software to include only footage of the target species. From this subset of video footage, an image dataset was created by extracting still video frames at regular intervals (for example, five frames for every one second of video). Although you can use almost any video editing software for this purpose, OpenCV (*Open Source Computer Vision Library (OpenCV)*, 2017) was used in this thesis for reproducibility and to retain the highest possible image quality. In total, 2015 frames (images) were selected for use in the dataset. This corresponds to 2541 unique instances of red roman.

5.2.2 Data Labelling

For maximum compatibility with a range of object recognition models, the video frames were annotated with instance segmentation masks – from these masks, minimum bounding boxes can easily be computed for general object detection tasks. VGG Image Annotator (VIA) (Dutta et al., 2016) from the Visual Geometry Group at the University of Oxford was used to label the dataset. VIA is free to download and does not require installation to be used. It is a standalone HTML file that works in most modern browsers (see Figure 5.2). It allows annotations to be saved in CSV, JSON or COCO JSON formats.

Masks (or polygon annotations) are in general more labour intensive than bounding boxes (rectangular annotations). Labelling sessions were timed for each participant (see Table 5.1). On average, a single object mask took 71.5 seconds to complete. (By comparison, it takes less than 5 seconds to draw a rectangular bounding box.) For the red roman dataset, this translates to a total of 51 hours, 36 minutes of labelling time (not including breaks).

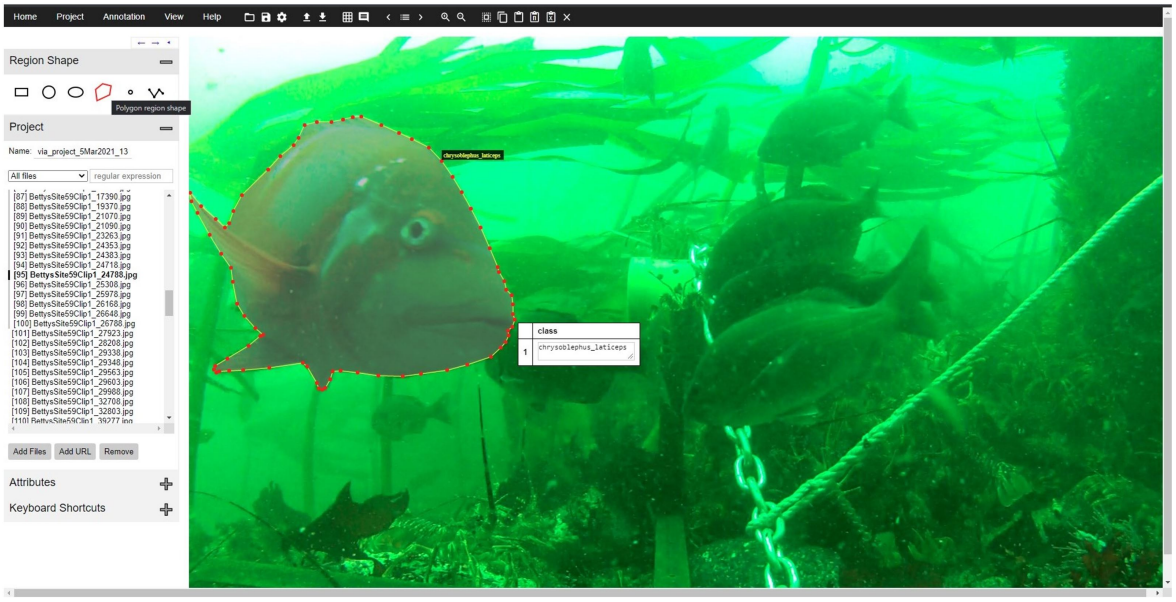

Figure 5.2: VGG Image Annotator (VIA) (Dutta et al., 2016) web application from the Visual Geometry Group at the University of Oxford. Here, a single red roman is labelled by hand with the polygon tool. Annotations are stored as a pixel-wise instance segmentation mask.

Table 5.1: Summary of timed labelling sessions. On average, a mask annotation took 71.5 seconds to complete.

Site Name	Annotations	per Annotation [s]	Total Duration [h]
Bettys Bay Site 10	224	76.7	4.8
Bettys Bay Site 28	151	75	3.1
Bettys Bay Site 56	397	83.1	9.2
Bettys Bay Site 59	593	74.7	12.3
Bettys Bay Site 68	189	75.4	4
Bettys Bay Site 83	152	73.4	3.1
Bettys Bay Site 87	258	75.4	5.4
Still Bay Site 1	130	61.3	2.2
Still Bay Site 2	246	65.3	4.5
Still Bay Site 3	201	55.1	3.1
	2541	**71.5**	**51.6**

5.2.3 Dataset Format

For the purposes of training, the red roman dataset was split into train, validation and test sets according to the 80:10:10 ratio that is common in literature (see Table 5.2). Since the captured footage for each geographical site may differ in water colour, clarity, lighting, etc., stratified sampling was used to keep frame (image) counts of all sites balanced across the train, validation, and test splits. However, a single frame can contain multiple annotations, and the count of annotations may differ per frame. Because of this, in the final split the number of annotations for different sites are actually unbalanced (search down column) but are still fairly consistent per site across the train, validation and test sets (search across row).

Table 5.2: Count (and proportion) of annotations per site across the training, validation and test dataset splits. The proportions (in %) are calculated column-wise, except for the total row (last row) which is calculated row-wise.

Site Name	Train	Validation	Test	All
Bettys Site 10	177 (8.7%)	22 (8.5%)	25 (10.1%)	224 (8.8%)
Bettys Site 28	123 (6%)	15 (5.8%)	13 (5.2%)	151 (5.9%)
Bettys Site 56	319 (15.7%)	38 (14.7%)	40 (16.1%)	397 (15.6%)
Bettys Site 59	473 (23.3%)	60 (23.2%)	60 (24.2%)	593 (23.3%)
Bettys Site 68	154 (7.6%)	19 (7.3%)	16 (6.5%)	189 (7.4%)
Bettys Site 83	127 (6.2%)	17 (6.6%)	8 (3.2%)	152 (6%)
Bettys Site 87	201 (9.9%)	26 (10%)	31 (12.5%)	258 (10.2%)
Still Bay Site 1	101 (5%)	15 (5.8%)	14 (5.6%)	130 (5.1%)
Still Bay Site 2	202 (9.9%)	27 (10.4%)	17 (6.9%)	246 (9.7%)
Still Bay Site 3	157 (7.7%)	20 (7.7%)	24 (9.7%)	201 (7.9%)
	2034 (80%)	259 (10.2%)	248 (9.8%)	2541 (100%)

5.3 Environment Setup

Modelling was performed on a local machine with specifications as given in Table 5.3. A conda environment was set up on the local machine as it is preferred for its compatibility with Windows to other methods available (such as Python's PIP) and can prevent conflicts that might arise from previously installed packages. A summary of

the environment is provided below in Table 5.4. The core programming language used is Python (v3.6.10).

Table 5.3: Specifications of the local machine used to perform modelling.

Specification	Description
OS	Windows 10 Home (64bit) (10.0.18363 Build 18363)
CPU	AMD Ryzen™7 3750H (2.3Ghz, up to 4.0Ghz)
GPU	RTX 2060 6GB Mobile (6GB GDDR6 Memory)
RAM	2 x 16GB DDR4 2400MHz (CLI14)
HDD	512GB NVMe SSD

Table 5.4: Summary of Python packages required.

Package	Version
numpy	1.19.0
scipy	1.5.1
Pillow	7.2.0
cython	0.29.21
matplotlib	3.2.2
scikit-image	0.17.2
tensorflow	1.5.0
keras	2.1.3
opencv-python	4.3.0.36
h5py	2.10.0
imgaug	0.4.0

5.4 Model

A popular and intuitive implementation of Mask R-CNN (He et al., 2017), is the Matterport (Abdulla, 2017) repository, and is the main library used in this thesis. The Matterport repository is written in Python and is built on the machine learning library TensorFlow (Abadi et al., 2016) developed by Google. The model generates bounding-box and segmentation mask predictions for every unique instance of an object within a larger image.

The Python script developed for this thesis sits "on top" of the Matterport library – in a way, the script acts as a high-level API to the Matterport Mask R-CNN implementation. It is available on the GitHub repository for this thesis (*Finding Nemo*, 2020), and will be hereinafter referred to as the "red roman model" for brevity.

In Mask R-CNN (He et al., 2017) the effective batch size is 16. This is achieved by using 8 GPUs in parallel with a mini-batch size set to 2 per GPU. The default configuration for the Matterport implementation is an effective batch size of 2 (1 GPU with a mini-batch size set to 2). The effective batch size for the red roman model was set to 1 (1 GPU with a mini-batch size set to 1). A single image can take up to 4GB of video memory, and the local machine is limited to 6GB of GDDR6 video memory, and therefore cannot accommodate a larger batch size.

The red roman model uses the smaller Resnet50 backbone network instead of the Resnet101. While this may decrease the accuracy of the red roman model, it significantly reduces the memory requirements while speeding up training. To further decrease the memory requirements of the model, the training images are resized such that their scale (shorter edge) is 460 pixels before being passed into the network.

In addition, before being presented to the network the images may undergo augmentation (see Figure 5.3). An image may be flipped on the horizontal (a) or vertical axis (b), or rotated within a maximum range of 90°(c). Since it is unlikely that the training data would be able to capture every position a fish may assume in practice, the artificial translation and rotation of portions of the training data helps prepare the model for what it may face on unseen data, and consequently increase the likelihood that the model will perform better (or generalise better) on unseen data.

Similarly, the brightness of an image may be increased or decreased (d) to simulate the leakage of light onto the body of a fish set against an extremely bright or dark background. An average blur may be applied to an image (e). This simulates the effect of a fast-moving fish captured between frames. Finally, random Gaussian noise may be applied to an image (f) to simulate the effect of tiny particulates that are suspended in varying intensity in bodies of water. In practice, a maximum of three augmentations may be applied to any one image.

Instead of setting the initialised weights to be randomised, a set of pre-trained weights trained on the MS COCO (Lin et al., 2014) dataset was used. This technique, known as transfer learning, is common when working with small datasets that may not contain sufficient training data. During training, the learning rate is set to 0.001 with

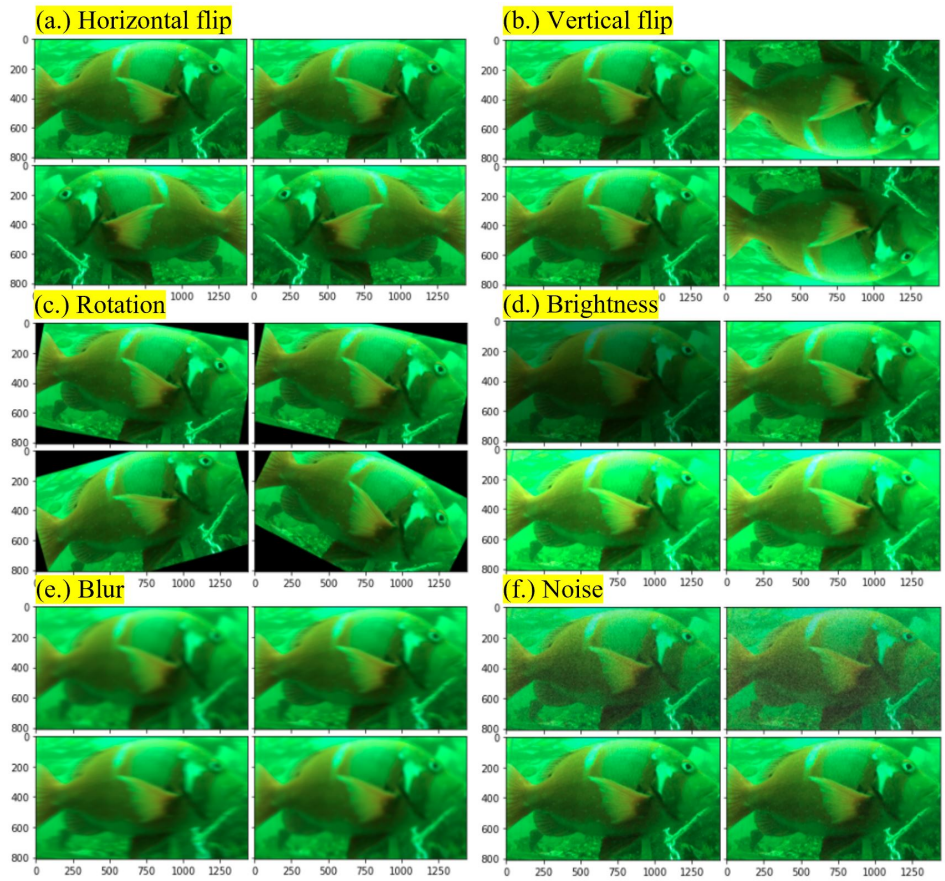

Figure 5.3: A mixture of data augmentation techniques applied to the red roman dataset: (*a*-*b*) Horizontal/vertical flip, (*c*) Rotation, (*d*) Brightness, (*e*) Average Blur, and (*f*) Gaussian Noise. (Best viewed on a high-resolution screen.)

a momentum of 0.9. The red roman model was evaluated using a separate Jupyter Notebook script also available on the companion Github repository (*Finding Nemo*, 2020).

The red roman model can be setup for inference on an image as well as on video. When setting video as an input to the model, the video is split into an ordered set of images (or frames). The frames are presented sequentially to the model for inference, before being stitched backed together. A generic centroid tracking method[1] was adapted and incorporated into the Matterport Mask R-CNN repository. In this way, the red roman model is in addition able to assign IDs to unique objects identified by the model, and the count and movement of these unique objects is stored and tracked.

[1]Source code available here: https://www.pyimagesearch.com/category/object-tracking

Chapter 6

Results and Discussion

This chapter discusses observations made during the training of the red roman instance segmentation model and reports the results achieved. The chapter concludes with a candid discussion of the limitations of this thesis and recommendations for further research.

Evaluation

The red roman model achieved a mAP_{50} of 80.35% and 81.45% on the validation and test set, respectively (see Table 6.1). At a stricter IoU threshold of 70%, the model achieved a mAP_{70} of 78.98% and 80.28% on the validation and test set, respectively. On the MS COCO mAP_{50-95} evaluation metric, which measures mAP averaged over IoU thresholds in $[0.50 : 0.05 : 0.95]$, the model achieved 62.52% and 63.83% on the validation and test set, respectively. A score above 80% on mAP_{50} on the test set is comparable with Mandal et al. (2018) and Li et al. (2015) and considered to be a good object recognition score[1].

Table 6.1: Results of the red roman Mask R-CNN model for the validation and test set, reported as mAP scores.

Dataset	mAP_{50}	mAP_{70}	mAP_{50-95}
Validation	80.35%	78.98%	62.52%
Test	81.45%	80.28%	63.83%

[1]It is not intended to make direct comparison with these studies, but rather to provide context for what is considered a good (or bad) mAP score.

The mAP score is the mean or the average of the AP scores across all images evaluated. For example, the average of the AP scores for every image in the test dataset at a threshold of 50% is the mAP_{50} score of 81.45% given in Table 6.1. Note that a subset of test images may perform worse than this average, and some may perform better. In effect, these act as outliers and warrant further inspection.

In Figure 6.1, the trained model reported AP_{50-95} scores of 82.5%, 72.5%, 85.0%, 90.0% for the images (*i.*), (*ii.*), (*iii.*), and (*iv.*), respectively. The AP scores of this subset of images significantly outperform the average for the test set to which they belong ($mAP_{50-95} = 63.83\%$). The model successfully detected and accurately mapped red roman that were facing away from the camera ((*ii.*), and (*iv.*)), were partially occluded (*iii.*), or were cut off within frame ((*i.*), and (*ii.*)) This is encouraging, since object detection is typically more challenging under these conditions.

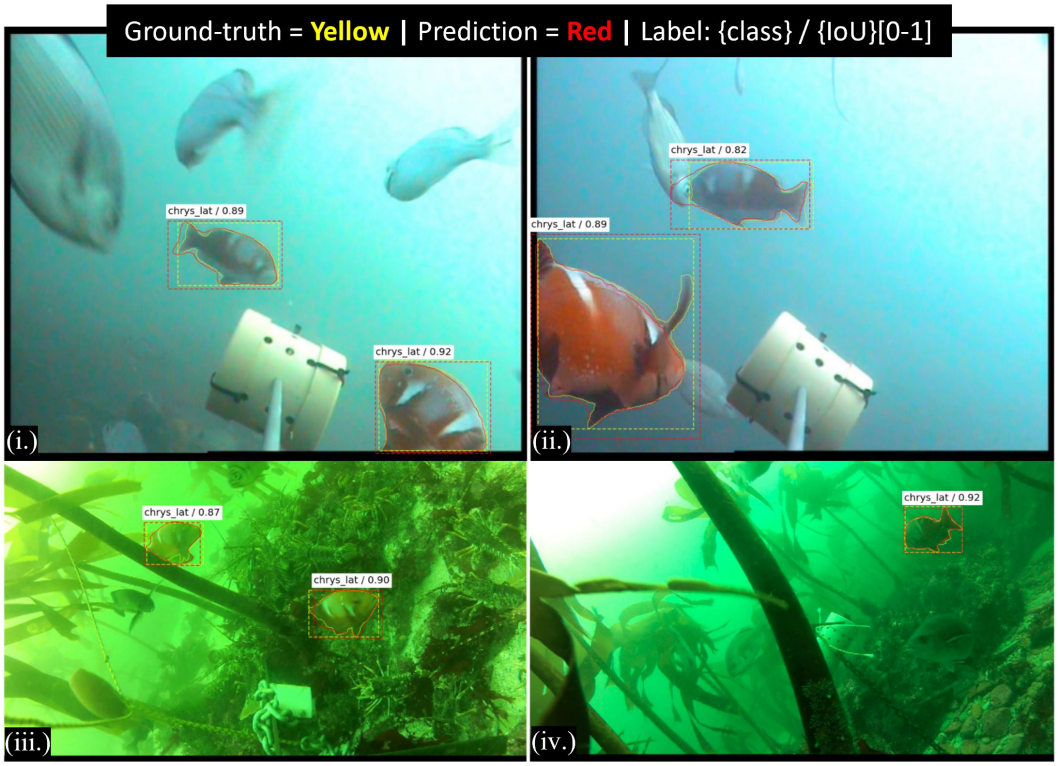

Figure 6.1: A random subset of test images for which the model performed significantly above the average for the overall test dataset ($AP_{subset} > mAP_{test}$).

In Figure 6.2, the trained model reported AP_{50-95} scores of 45.5%, 58.8%, 40.0%, 40.0% for the images (*i.*), (*ii.*), (*iii.*), and (*iv.*), respectively. The AP scores of this subset of images significantly underperform the average for the test set to which they

belong ($mAP_{50-95} = 63.83\%$). While the model was able to successfully differentiate individual red roman from a clustered group in some cases, it did not do so consistently. It frequently grouped one or more individuals into a single prediction. For this subset of images even though the IoU scores for individual predictions were high, the images scored AP_{50-95} values that were below average as a result of failing to predict entire individual red roman – an increase in the number of false negatives contributes to a lower recall, and consequently lower overall AP scores. The model seems to perform comparatively worse when localising red roman that are clustered in groups ((*i.*) and, (*ii.*)), are relatively far away from the camera and are therefore smaller in scale ((*iii.*), and (*iv.*)), or are set against backgrounds of intense brightness where light leaks onto the body of the fish (*ii.*). The performance of the model in these situations may be improved with heavier augmentation (especially to simulate the scenarios above for which the model is not robust to), or by increasing the amount of training data to better reflect these conditions.

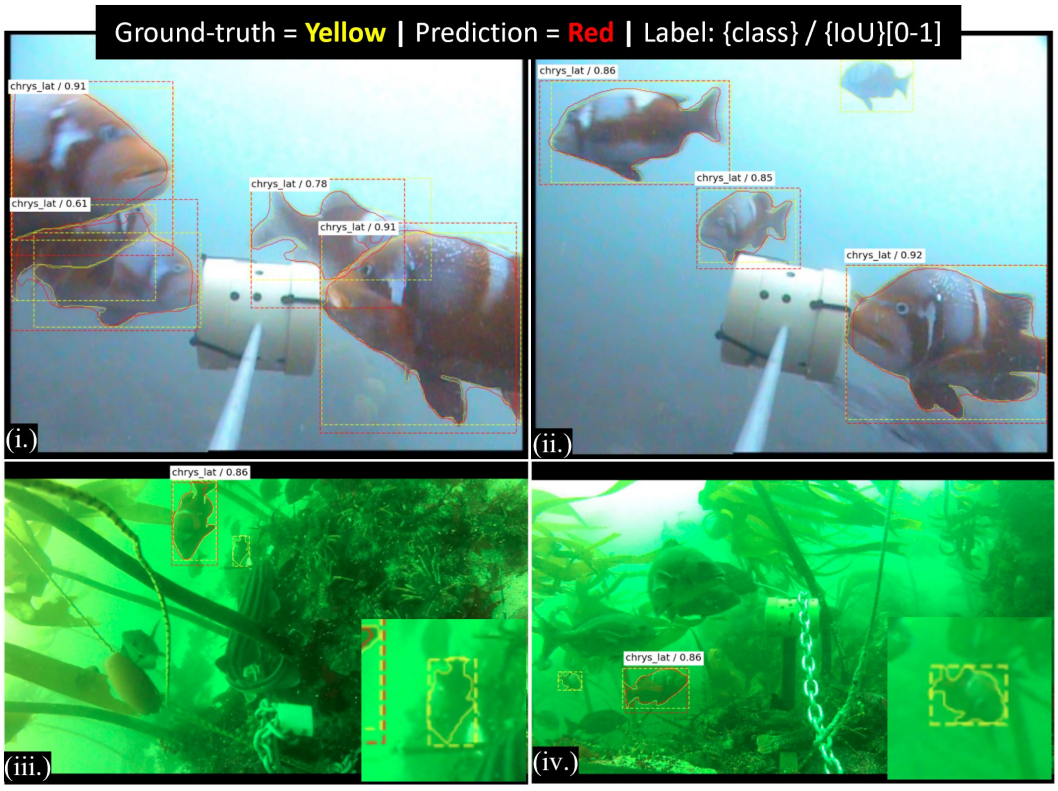

Figure 6.2: A random subset of test images for which the model performed significantly below the average for the overall test dataset ($AP_{subset} < mAP_{test}$).

Training

Figure 6.3 plots the loss function of the red roman model against the training epoch. The loss function (in this case, log loss) is evaluated on both the training set and the evaluation set for every epoch. A 5-point moving average is used to smooth both learning curves – this can help make trends in the data stand out more clearly.

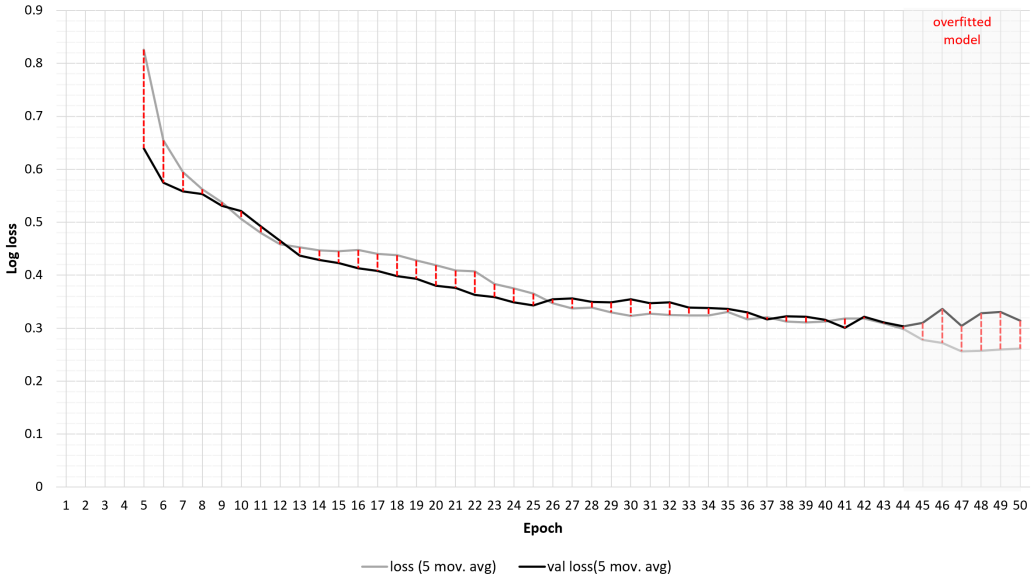

Figure 6.3: Learning curve: training and validation log loss for the red roman model evaluated at each training epoch.

It is observed that the training and validation error decrease sharply initially, but that this rate of learning becomes more gradual as training progresses. Throughout training, the training error continues to decrease while the validation error starts to level-out after approximately 40 epochs. Here, the gap between the training and validation error starts to widen. This indicates that the model has now started to overfit on the training data. That is, the model has started to fit specific features in the training set that do not generalise well on the validation set. To continue training past this point would result in a model that has overfit on the training data. Consequently, the model would be expected to outperform on the data it has seen, but perform relatively poorly when tested on new or unseen data. Such a model is not robust to real world situations where models should be able to generalise well on incoming streams of new (but familiar) data.

It should be noted that the log loss on the validation set does not significantly

increase at any point during training. The training and validation error are within close range of each other for the majority of the training schedule. Regularisation techniques, such as the data augmentation methods described in Chapter 3 would have played a role here in preventing excessive model overfitting.

It is useful to take a closer look at the validation error during training shown in Figure 6.3. This error, a multi-task log loss, comprises the class, bounding-box and mask log loss, as defined in Chapter 4, and is decomposed and plotted against the training epoch in Figure 6.4. It is observed during training that $\mathcal{L}_{bbox}$ contributes the most to the overall validation error. Furthermore, of the three defined task losses, $\mathcal{L}_{bbox}$ is the most volatile while $\mathcal{L}_{cls}$ and $\mathcal{L}_{mask}$ achieve lower log loss scores and are more predictable (smooth) over training.

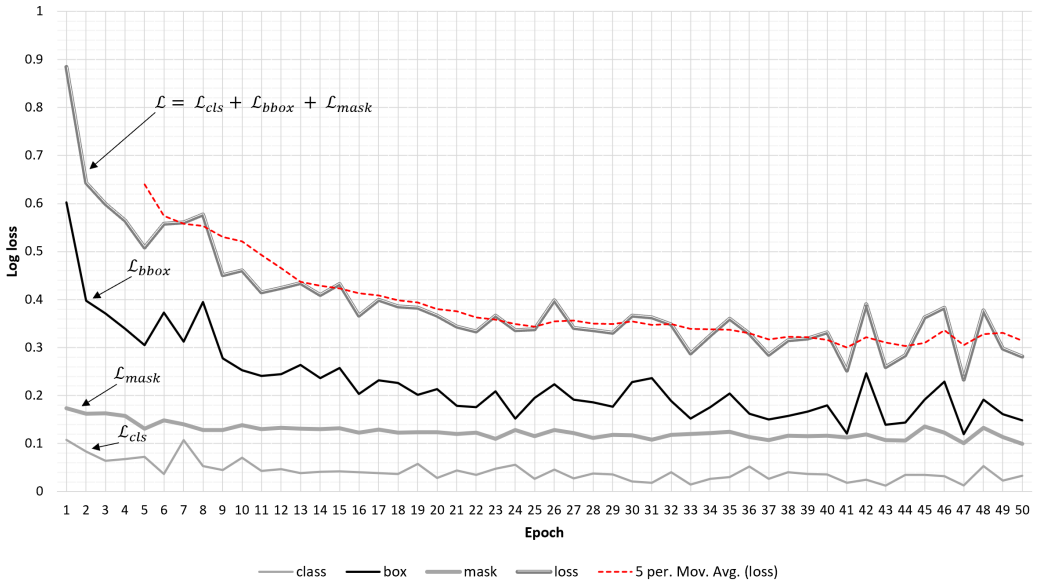

Figure 6.4: Learning curve: multi-task loss comprised of the class, bounding-box and mask log loss for the red roman model on the validation set.

Red roman, like most fish, are highly deformable objects – a fish can twist and tuck its body and retract or extend its fins in three-dimensional space. Figure 6.5 posits how the angle of the camera in respect to the fish, and the extent of deformation, can lead to a large discrepancy between bounding-box and mask predictions. Predicted segmentation masks tend to hug an object, cutting off – or smoothing over – small protrusions that may project from the main body of the object (like the fin of a fish when viewed head-on). This is turn produces a shortened minimum bounding box.

In contrast, the ground-truth masks annotated by a human expert can be extremely accurate and sensitive to even the smallest protrusion (like a pectoral fin), and this in turn produces an extended minimum bounding box. The result is a significant delta on overlapping bounding boxes (red), for what may be considered an insignificant delta on overlapping masks (yellow) (see Figure 6.5). This may be partly responsible for the higher (and more volatile) $\mathcal{L}_{bbox}$ task loss when compared to $\mathcal{L}_{mask}$ on the validation set.

Figure 6.5: The predicted mask (red) for this individual red roman cuts off the pectoral fin, while the ground-truth mask (yellow) accurately maps the pectoral fin. The mask delta (yellow highlighted) is insignificant, but results in a large delta between overlapping bounding boxes (red highlighted).

Tracking

An example of a video submitted to the model appears in Figure 6.6. The presence of a red roman is marked by a blue instance segmentation mask and a blue minimum bounding box. The centre of the minimum bounding box (centroid) is marked with a blue dot. Using a generic centroid tracking method, each unique instance of a red roman is assigned an ID, and the position[2] of the red roman is stored for every frame. In this way, the movement of all unique red roman identified by the model can be tracked. In addition, the count of red roman by frame – or at a discrete timestamp – is known.

[2]The position of an object is stored as the x-y co-ordinate pair of the centroid of the object.

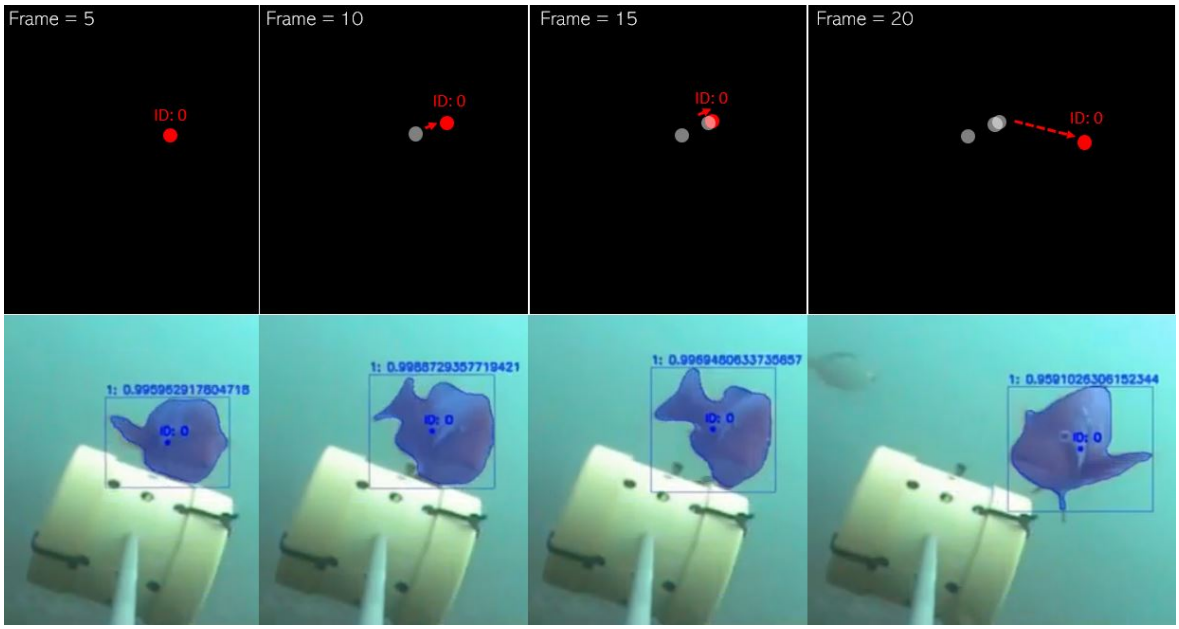

Figure 6.6: An example of a video sequence evaluated by the red roman model. The count and position of all unique objects is stored for each frame. The movement (or change in position) of objects across frames, as well as the count of identified objects per frame is easily computed.

The following chapter concludes with the contributions of this thesis, the limitations and further work.

Chapter 7

Conclusion

Object recognition within unconstrained underwater environments presents a diverse set of challenges for researchers. The most obvious has to do with the nature of water itself, as Akkaynak & Treibitz (2019) eloquently states: "An underwater photo is the equivalent of one taken in air, but covered in thick, colored fog, subject to an illuminant whose white point and intensity changes as a function of distance." Moreover, objects underwater are free to move within a large three-dimensional space and are consequently often subject to extreme variation in translation, rotation, and scale. Fish, in particular, are able to twist and contort the shape of their body – such extreme deformation to the shape of an object is a common problem in developing state-of-the-art object detectors for animals (Chen et al., 2014).

Despite these challenges, the need for greater automation in the observation of underwater marine environments is evident: on average, a human expert spends over two hours identifying and counting fish in a video that is one hour long (Williams et al., 2012). The manual analysis of marine data is both labour-intensive and extremely costly (Weinstein, 2018*b*). For scientific study to be scalable, the rate of data analysis must be at least equal to the rate of acquisition, yet for two of the largest marine datasets in existence less than 5% of data will ever be analysed by experts (Moniruzzaman et al., 2017). It is evident that an automated method of analysis offering a comparable accuracy at a fraction of the time and cost is critical and should be considered a research priority.

The contributions of this thesis are: a novel image dataset of red roman (*Chrysoblephus laticeps*) that can be used to train object recognition models; and, a Mask R-CNN framework that achieves accurate localisation, classification, counting and tracking of

red roman in unconstrained underwater environments.

The red roman dataset comprises 2015 images and 2541 unique instance segmentation masks and is backwards compatible with object detection as well as image classification models. It took a minimum of 51 hours to prepare and label the dataset, which is free to download and use for academic purposes.

The red roman Mask R-CNN model was trained on the aforementioned dataset. Instead of a random set of initialised weights, a set of pre-trained weights trained on the MS COCO (Lin et al., 2014) dataset was used. This technique, known as transfer learning, is common when working with small datasets that may not contain sufficient training data. A known limitation of the red roman dataset is the relatively small number of annotations versus some other publications (Ditria et al. (2020), for example). Data augmentation techniques were used to artificially increase the amount of training data available to the model. The model was trained for a total of 50 epochs and achieved a mAP_{70} accuracy of 78.98% on the validation dataset, and a mAP_{70} accuracy of 80.28% on the test dataset. A mAP_{70} is considered as reasonably strict since more than 70% of the prediction must overlap with the ground-truth (in contrast to just 50% that might be considered more lenient). The fact that the model performs well on previously unseen data suggests that it is capable of generalising to new streams of data not included in this research. This is critical for the general usability of the model in a practical setting.

It is hoped that the red roman dataset created in this thesis eases the burden of computer vision research within South Africa by decoupling data collection from machine learning research efforts – especially within the context of underwater marine object detection, for which there is a distinct lack in South African publications.

The red roman Mask R-CNN model developed in this thesis will assist marine research efforts where accurate, cheap and fast automated methods of video analysis of marine environments are necessary. The model will serve as a proof-of-concept that machine learning based methods of locating, identifying, counting and tracking fish in underwater video can replace or at least supplement human efforts. Given the backlog of marine video data, and a growing trend of increased data collection, this model offers the opportunity for quick, robust and independently reproducible scientific study of marine biodiversity.

7.1 Limitations

The red roman dataset comprises 2015 images and 2541 unique annotations for a single object class. Ditria et al. (2020) demonstrate that the performance of their object segmentation model was positively correlated to the size of the training set. Predictably, the accuracy of the model (scored by mAP_{50} on a validation dataset) improved as the number of annotations increased. The results of Ditria et al. (2020) is particularly significant to this thesis as there are some notable similarities, such as the choice to limit the focus of object detection to a single fish species (luderick), and the use of footage of luderick in unconstrained underwater environments. It is posited that increasing the size of the training set from 2541 annotations to 6000 annotations as in Ditria et al. (2020) will likely increase the performance capability of the red roman model.

Training was conducted on a local machine with limited resources. This is in contrast to the seemingly unlimited computing resources offered by various on-demand cloud computing platforms that are able to scale to meet research needs. In He et al. (2017) for example, a machine equipped with 8 GPUs each with a minimum of 8GB of video RAM was used[1]. In this research, a single GPU with 6GB of video memory was used.

Moreover, the red roman model uses the smaller Resnet50 backbone network instead of the Resnet101. While this may decrease the accuracy of the red roman model, it significantly reduces the memory requirements while speeding up training. To further decrease the memory requirements of the model, the training images are resized such that their scale (shorter edge) is 460 pixels before being passed into the network.

The choice of the resolution of input images, Mask R-CNN network backbone, batch size, number of training epochs, and other configurations were inevitable trade-offs between accuracy and feasibility. It is recommended in the future to use a cloud computing platform so that these architecture choices can be made based on the empirical evidence of training-validation curves (as they should be), and not on the basis of constrained resources.

[1]The authors do not specify the exact model or available memory of each GPU. However, they do state that each video card was used to process a mini-batch of two images (of 800 pixels each at shorter edge). This roughly translates to about 4GB of necessary video memory per image, for a total of 8GB per GPU.

7.2 Further Research

Semi-supervised learning, a method that utilises a small set of labelled data and a large set of unlabelled data, has shown to increase the accuracy of classification models where only limited data is available, and to be less prone to overfitting on this data (Erhan et al., 2010). Given the labour-intensive and often costly process of manually labelling data, semi-supervised learning offers a viable alternative (or addition) to supervised learning techniques. These techniques should be particularly useful in South Africa where large, good quality datasets are absent in literature.

Akkaynak & Treibitz (2019) notes the challenges of developing state-of-the-art object recognition models in the underwater domain. Learning-based methods struggle, in part, due to the inevitable degradation of image quality due to light reflected from particles suspended in the water column. They present a method called *Sea-thru* for the robust recovery of lost colours in underwater imagery, and show that it outperforms existing methods that are either unstable, too sensitive or only work for short object ranges (Akkaynak & Treibitz, 2019). Sea-thru is designed to work on RGBD images (those that contain depth information). The vast majority of image datasets, however, including the one used in this research, use the popular JPEG compression standard, since it offers a desirable trade-off between storage size and image quality. An image that goes through JPEG compression becomes an artifact of the original image and no longer contains the raw information captured by a camera's multiple sensors that is needed by Sea-thru. Despite this, the ability of Sea-thru to "remove the water from underwater imagery" opens up a new research opportunity to explore the full range of computer vision algorithms that have otherwise performed sub-optimally in the underwater domain.

Bibliography

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M. et al. (2016), Tensorflow: A system for large-scale machine learning, *in* '12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)', pp. 265–283.

Abdulla, W. (2017), 'Mask r-cnn for object detection and instance segmentation on keras and tensorflow', https://github.com/matterport/Mask_RCNN.

Akkaynak, D. & Treibitz, T. (2019), Sea-thru: A method for removing water from underwater images, *in* 'Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition', pp. 1682–1691.

Boom, B. J., Huang, P. X., Beyan, C., Spampinato, C., Palazzo, S., He, J., Beauxis-Aussalet, E., Lin, S.-I., Chou, H.-M., Nadarajan, G. et al. (2012), 'Long-term underwater camera surveillance for monitoring and analysis of fish populations', *VAIB12* .

Chen, X., Mottaghi, R., Liu, X., Fidler, S., Urtasun, R. & Yuille, A. (2014), Detect what you can: Detecting and representing objects using holistic models and body parts, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 1971–1978.

Dalal, N. & Triggs, B. (2005), Histograms of oriented gradients for human detection, *in* '2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)', Vol. 1, IEEE, pp. 886–893.

Dando, J. W. (2020), A baited remote underwater video survey of the goukamma marine protected area's ichthyofauna and a subsequent community structure compari-

son with the betty's bay, stilbaai, and tsitsikamma marine protected areas, Master's thesis, Faculty of Science.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K. & Fei-Fei, L. (2009), Imagenet: A large-scale hierarchical image database, *in* '2009 IEEE conference on computer vision and pattern recognition', Ieee, pp. 248–255.

Ditria, E. M., Lopez-Marcano, S., Sievers, M., Jinks, E. L., Brown, C. J. & Connolly, R. M. (2020), 'Automating the analysis of fish abundance using object detection: Optimizing animal ecology with deep learning', *Frontiers in Marine Science* **7**, 429.

Dutta, A., Gupta, A. & Zissermann, A. (2016), 'Vgg image annotator (via)', http://www.robots.ox.ac.uk/~vgg/software/via/.

Erhan, D., Courville, A., Bengio, Y. & Vincent, P. (2010), Why does unsupervised pre-training help deep learning?, *in* 'Proceedings of the thirteenth international conference on artificial intelligence and statistics', JMLR Workshop and Conference Proceedings, pp. 201–208.

Everingham, M., Van Gool, L., Williams, C. K., Winn, J. & Zisserman, A. (2010), 'The pascal visual object classes (voc) challenge', *International journal of computer vision* **88**(2), 303–338.

Everingham, M. & Winn, J. (2011), 'The pascal visual object classes challenge 2012 (voc2012) development kit', *Pattern Analysis, Statistical Modelling and Computational Learning, Tech. Rep* **8**.

Fei-Fei, L., Fergus, R. & Perona, P. (2004), Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories, *in* '2004 conference on computer vision and pattern recognition workshop', IEEE, pp. 178–178.

Finding Nemo (2020), https://github.com/FishCV/fishcv.github.io.

Fletcher, S., Islam, M. Z. et al. (2018), 'Comparing sets of patterns with the jaccard index', *Australasian Journal of Information Systems* **22**.

Geiger, A., Lenz, P., Stiller, C. & Urtasun, R. (2013), 'Vision meets robotics: The kitti dataset', *The International Journal of Robotics Research* **32**(11), 1231–1237.

Girshick, R. (2015), Fast r-cnn, *in* 'Proceedings of the IEEE international conference on computer vision', pp. 1440–1448.

Girshick, R., Donahue, J., Darrell, T. & Malik, J. (2014), Rich feature hierarchies for accurate object detection and semantic segmentation, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 580–587.

Goodfellow, I., Bengio, Y., Courville, A. & Bengio, Y. (2016), *Deep learning*, Vol. 1, MIT press Cambridge.

Griffin, M. A., Neal, A. & Parker, S. K. (2007), 'A new model of work role performance: Positive behavior in uncertain and interdependent contexts', *Academy of management journal* **50**(2), 327–347.

He, K., Gkioxari, G., Dollár, P. & Girshick, R. (2017), Mask r-cnn, *in* 'Proceedings of the IEEE international conference on computer vision', pp. 2961–2969.

Hinton, G. E., Osindero, S. & Teh, Y.-W. (2006), 'A fast learning algorithm for deep belief nets', *Neural computation* **18**(7), 1527–1554.

ImageCLEF (2014), *LifeCLEF Subtask 1 - Video Based Fish Identification.*
URL: *https://www.imageclef.org/2014/lifeclef/fish*

Knausgård, K. M., Wiklund, A., Sørdalen, T. K., Halvorsen, K., Kleiven, A. R., Jiao, L. & Goodwin, M. (2020), 'Temperate fish detection and classification: a deep learning based approach', *arXiv preprint arXiv:2005.07518* .

Krizhevsky, A., Sutskever, I. & Hinton, G. E. (2012), 'Imagenet classification with deep convolutional neural networks', *Advances in neural information processing systems* **25**, 1097–1105.

Learned-Miller, E., Huang, G. B., RoyChowdhury, A., Li, H. & Hua, G. (2016), Labeled faces in the wild: A survey, *in* 'Advances in face detection and facial image analysis', Springer, pp. 189–248.

LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P. (1998), 'Gradient-based learning applied to document recognition', *Proceedings of the IEEE* **86**(11), 2278–2324.

Li, X., Shang, M., Qin, H. & Chen, L. (2015), Fast accurate fish detection and recognition of underwater images with fast r-cnn, *in* 'OCEANS 2015-MTS/IEEE Washington', IEEE, pp. 1–5.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P. & Zitnick, C. L. (2014), Microsoft coco: Common objects in context, *in* 'European conference on computer vision', Springer, pp. 740–755.

Lowe, D. G. (1999), Object recognition from local scale-invariant features, *in* 'Proceedings of the seventh IEEE international conference on computer vision', Vol. 2, Ieee, pp. 1150–1157.

Mandal, R., Connolly, R. M., Schlacher, T. A. & Stantic, B. (2018), Assessing fish abundance from underwater video using deep neural networks, *in* '2018 International Joint Conference on Neural Networks (IJCNN)', IEEE, pp. 1–6.

Moniruzzaman, M., Islam, S. M. S., Bennamoun, M. & Lavery, P. (2017), Deep learning on underwater marine object detection: A survey, *in* 'International Conference on Advanced Concepts for Intelligent Vision Systems', Springer, pp. 150–160.

Open Source Computer Vision Library (OpenCV) (2017), https://github.com/itseez/opencv.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L. et al. (2019), 'Pytorch: An imperative style, high-performance deep learning library', *arXiv preprint arXiv:1912.01703* .

Pimm, S. L., Alibhai, S., Bergl, R., Dehgan, A., Giri, C., Jewell, Z., Joppa, L., Kays, R. & Loarie, S. (2015), 'Emerging technologies to conserve biodiversity', *Trends in ecology & evolution* **30**(11), 685–696.

Qin, H., Li, X., Liang, J., Peng, Y. & Zhang, C. (2016), 'Deepfish: Accurate underwater live fish recognition with a deep architecture', *Neurocomputing* **187**, 49–58.

Ren, S., He, K., Girshick, R. & Sun, J. (2015), Faster r-cnn: Towards real-time object detection with region proposal networks, *in* 'Advances in neural information processing systems', pp. 91–99.

Rosebrock, A. (2019*a*), *Deep learning for computer vision with Python: Bonus Bundle*, PyImageSearch.

Rosebrock, A. (2019*b*), *Deep learning for computer vision with Python: ImageNet Bundle*, PyImageSearch.

Rosebrock, A. (2019*c*), *Deep learning for computer vision with Python: Practitioner Bundle*, PyImageSearch.

Rosebrock, A. (2019*d*), *Deep learning for computer vision with Python: Starter Bundle*, PyImageSearch.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M. et al. (2015), 'Imagenet large scale visual recognition challenge', *International journal of computer vision* **115**(3), 211–252.

Salman, A., Siddiqui, S. A., Shafait, F., Mian, A., Shortis, M. R., Khurshid, K., Ulges, A. & Schwanecke, U. (2020), 'Automatic fish detection in underwater videos by a deep neural network-based hybrid motion learning system', *ICES Journal of Marine Science* **77**(4), 1295–1307.

Santurkar, S., Tsipras, D., Ilyas, A. & Madry, A. (2018), 'How does batch normalization help optimization?', *arXiv preprint arXiv:1805.11604* .

Simonyan, K. & Zisserman, A. (2014), 'Very deep convolutional networks for large-scale image recognition', *arXiv preprint arXiv:1409.1556* .

Springenberg, J. T., Dosovitskiy, A., Brox, T. & Riedmiller, M. (2014), 'Striving for simplicity: The all convolutional net', *arXiv preprint arXiv:1412.6806* .

Sun, X., Shi, J., Liu, L., Dong, J., Plant, C., Wang, X. & Zhou, H. (2018), 'Transferring deep knowledge for object recognition in low-quality underwater videos', *Neurocomputing* **275**, 897–908.

Torralba, A., Fergus, R. & Freeman, W. T. (2008), '80 million tiny images: A large data set for nonparametric object and scene recognition', *IEEE transactions on pattern analysis and machine intelligence* **30**(11), 1958–1970.

Villon, S., Mouillot, D., Chaumont, M., Darling, E. S., Subsol, G., Claverie, T. & Villéger, S. (2018), 'A deep learning method for accurate and fast identification of coral reef fishes in underwater images', *Ecological Informatics* **48**, 238–244.

Viola, P. & Jones, M. (2001), 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', Vol. 1, IEEE, pp. I–I.

Wang, N., Wang, Y. & Er, M. J. (2020), 'Review on deep learning techniques for marine object recognition: Architectures and algorithms', *Control Engineering Practice* p. 104458.

Wang, X., Ouyang, J., Li, D. & Zhang, G. (2019), 'Underwater object recognition based on deep encoding-decoding network', *Journal of Ocean University of China* **18**(2), 376–382.

Weinstein, B. G. (2018*a*), 'A computer vision for animal ecology', *Journal of Animal Ecology* **87**(3), 533–545.

Weinstein, B. G. (2018*b*), 'Scene-specific convolutional neural networks for video-based biodiversity detection', *Methods in Ecology and Evolution* **9**(6), 1435–1441.

Williams, K., Rooper, C. N. & Harms, J. J. H. (2012), 'Report of the national marine fisheries service automated image processing workshop, september 4-7, 2012, seattle, washington'.

Zitnick, C. L. & Dollár, P. (2014), Edge boxes: Locating object proposals from edges, *in* 'European conference on computer vision', Springer, pp. 391–405.