Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Sondre Aleksander Bergum

Object detection and instance segmentation of planktonic organisms using Mask R-CNN for real-time in-situ image processing.

An AILARON Project

Master's thesis in Cybernetics and Robotics
Supervisor: Annette Stahl, Aya Saad
July 2020

NTNU
Norwegian University of Science and Technology

Sondre Aleksander Bergum

Object detection and instance segmentation of planktonic organisms using Mask R-CNN for real-time in-situ image processing.

An AILARON Project

Master's thesis in Cybernetics and Robotics
Supervisor: Annette Stahl, Aya Saad
July 2020

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Abstract

This thesis is an investigation of recent state-of-the-art methods and architectures for segmentation[61, 38, 37, 78, 49] using Facebook Artificial Intelligence Research's (FAIR) software framework Detectron2's[77] implementation of Mask R-CNN[38]. We validate the method's results through experiments over the MS COCO[48] dataset as reported by He et al., and we conduct our own training and evaluation of several different configurations of the method on our own data. For this we provide a novel custom dataset[14] from planktonic images captured in-situ[60] in a lab environment suited for object detection and instance segmentation. We provide results, trained models, and the code necessary to embed a module of the Mask R-CNN implementation into existing in-situ imaging systems. Our results show that the method performs excellently in terms of accuracy while having a low enough computational overhead to operate in real-time in-situ.

Code and miscellaneous files have been made available at: `https://github.com/AILARON/Segmentation`

Sammendrag

Denne oppgaven er en undersøkelse av nylig presenterte state-of-the-art metoder og nettverksarkitekturer for bildesegmentering[61, 38, 37, 78, 49] ved bruk av Facebook Artificial Intelligence Research (FAIR) sitt software rammeverk, Detectron2 sin implementasjon av Mask R-CNN[38]. Vi bekrefter resultatene fra metoden presentert av He et al. gjennom våre egne ekperimenter utført på MS COCO[48] datasettet. Vi utfører også vår egen trening og evaluering av flere forskjellige konfigurasjoner av metoden på egen data. For dette tilfører vi et nytt, egenprodusert, annotert datasett[14] bestående av plankton bilder tatt in-situ[60] i lab-omgivelser, som egner seg for objekt deteksjon og instansiert segmentering. Vi oppgir resultater, ferdig trente modeller og kode nødvendig for å innlemme en modul av Mask R-CNN implementasjonen i eksisterende in-situ avbildningssystermer. Resultatene våre viser at metoden utfører oppgaven med strålende nøyaktighet samtidig som at operasjonstiden er lav nok til å prosessere data i sanntid in-situ.

Kode og andre filer er tilgjengeliggjort ved:
`https://github.com/AILARON/Segmentation`

Contents

List of Tables

List of Figures

Preface

This master's thesis is submitted as a part of the requirements for the master's degree at the Department of Engineering Cybernetics at the Norwegian University of Science and Technology.

Acknowledgment

The work presented in this thesis has been carried out under the supervision of Associate Professor Annette Stahl and Postdoctoral Fellow Aya Saad at the Department of Engineering Cybernetics, NTNU. The research is part of the AILARON project[9] which is funded by RCN FRINATEK IKTPLUSS program (project number 262701) and supported by NTNU AMOS.

As a result of the thesis we have submitted an abstract to the OCEANS 2020 Gulf Coast conference, see appendix A. Since the submission of this abstract the focus of the work has shifted slightly, so it doesn't necessarily correspond to the work finished at the time of the submission of this thesis. If the abstract is accepted, I will of course continue the work in order to meet the promised results.

This master's thesis is a continuation of a specialization project I conducted during the autumn of 2019. As is customary, the specialization project is not published. This means that important background theory and methods from the project report may be restated in full throughout this report to keep the thesis self-contained and provide the best reading experience. This mainly applies to chapter 2 which is mostly repetition,

but smaller parts throughout the rest of the thesis may also be repetition. The original project problem description published by Annette Stahl included references to the work of Badrinarayanan et al. (2017)[11], Garcia-Garcia et al. (2017)[29], Gupta et al. (2014)[32] and Hariharan et al. (2014)[35].

During the project, I have received guidance and feedback from both of my supervisors at-need and in regular meetings. From the 12th of March throughout the rest of the thesis period this guidance has taken place in the form of digital communication.

I have been provided with access to multiple tools through my supervisors. I was provided with remote access to a lab computer, and the ability to work directly on it if I should choose to do so. The computer is equipped with hardware necessary to perform the heavy processing deep learning methods often entail. The following are the specifications of the lab computer:

- OS: Ubuntu 18.04.2 LTS
- CPU: Intel LGA1151 i9 - 9900K
- GPU: 2x ASUS RTX2080Ti Turbo
- RAM: 64 GB
- SSD: Crucial MX500 2TB
- HDD: Seagate Skyhawk 6TB

Covid-19

Because of the need to access the hardware remotely and the transfer speed supported by the NAS drive mounted in the computer where the data was stored, extensive use of remote desktop has been necessary to reduce the need for file transfers to inspect the experiment results in images. Because of the lack of access to the on-site infrastructure on campus due to infection control measures, the performance and reliance of this remote access have been significantly limited. The issues caused by this and the time taken to explore and establish alternative methods to conduct the work as well as the

extra time necessary to perform the tasks without suitable alternatives has had a huge impact on the progress of the work completed in this thesis. To remedy some of the effects of this situation some extra time was allotted to the project period.

Implied or omitted details

The work of facilitating the use of software systems necessary for the experiments and processes detailed in this thesis will not be discussed further in the later parts of this thises. This means for instance setting up virtual environments with the correct software packages and dependencies, the study of frameworks in order to be able to use and further develop the implementations or debugging of faulty systems. Although these processes can take a considerable amount of time they are omitted in later chapters as they are considered implied and not directly contributing to the results presented. Failed or abandoned efforts will be discussed briefly in this section.

A sufficient amount of data captured during the field missions by the LAUV has already been processed and made available on the hardware to our disposal. This has significantly reduced the necessity of making sure I can run the processing framework myself. During the earlier stages of the project, an effort of familiarizing myself sufficiently to use, alter, and further develop this system was conducted. This was on a deprecated version of the system that has since been updated. After unsuccessful attempts in conjunction with my supervisor, the focus of the work gravitated away from the existing system towards other parts of the work presented in this thesis as further efforts were not considered worth the time investment. Other unfinished or failed efforts include one method in particular called DANet by Jun Fu (2019)[43]. This approach embodied one of the ambitions of this project which was to apply attention block modules to the segmentation. DANet seemed very interesting, but was abandoned due to the inability reproduce the results presented by the litterature.

Unless otherwise stated, all figures and illustrations have been created by the author.

Sondre Aleksander Bergum
Trondheim, July 2020

Chapter 1

Introduction

This introductory chapter will briefly provide context for the material presented in this report and give motivation and a description of the problem to be solved. The contributions of the report are defined and elaborated and we provide a map for the reader outlining the rest of this thesis.

1.1 Motivation and Problem description

Planktonic organisms form the principal source of food for consumers on higher trophic levels in the food chain. These organisms are susceptible to environmental changes, and studying their temporal variation in spatial abundance and taxa distribution[24] plays an integral part in understanding and predicting the development of ecosystems in the ocean. Manual methods for analyzing the gathered data are time-consuming and limits the study in the research field.

Currently, research[68, 16, 80, 18, 66] focuses on applying handcrafted computer vision techniques for performing automatic visual recognition tasks on planktonic images to aid scientists in their work. There has been a lot of progress in the tasks of using deep learning for detection, classification, and segmentation of images recently [51, 70, 37, 38]. State-of-the-art techniques are being applied to autonomous driving,

general scene understanding, medical imagery, and inspection tasks in industrial applications, among other things. Efforts of applying deep learning methods to plankton imaging systems have mainly been applied to classification, while detection and segmentation have been left to traditional methods in these types of frameworks. We seek to apply recent deep learning techniques for segmentation to accurately identify and extract plankton from real-time time-series image scenes taken in-situ.

1.2 Contributions

We showcase that recent state-of-the-art methods of deep learning applied to the visual recognition tasks of object detection and instance segmentation on planktonic data produce excellent results in terms of accuracy while having a low enough computational overhead to operate in real-time in-situ. The main contribution of this thesis is: due to the lack of annotated planktonic images suited for instance segmentation, we provide such a manually annotated dataset on the widely accepted format standard of MS COCO[48]. We show the performance of Mask R-CNN's[38] detection and instance segmentation trained on the novel custom data and validate its viability and superiority in replacing traditional methods. We provide results, trained models, and code necessary to embed a module of this method in existing in-situ imaging systems.

1.3 Thesis Outline

The rest of this thesis is organized as follows. Chapter 2 summarizes existing necessary and relevant knowledge on deep learning and image segmentation. It covers significant contributions and recent developments to the field leading up to the state-of-the-art. Chapter 3 elaborate on the details of the methods and implementations used for the experiments in this thesis while chapter 4 describes the datasets evaluated by them. Definitions of evaluation metrics and what they represent can be found in chapter 5, before we present our experimental results in terms of these metrics in chapter 6. In chapter 7 we discuss what can be gathered from the results while chapter 8 is devoted to the conclusion and proposals for future work.

Chapter 2

Theory Background

In this chapter, the necessary and relevant knowledge to follow and fully understand the problem and possible solutions to it is discussed. First computer vision and its role in deep learning will be described as well as how the state-of-the-art methods for image processing have developed. The following will be an introduction to the problem of segmentation and how the previously mentioned methods evolved into the most promising concepts used for segmentation today.

2.1 Computer vision

The subfield of single-image processing can be further divided into the tasks of **Classification**, **Detection**, **Localization** and **Segmentation** of objects. These different tasks are not isolated from one another. They can be viewed as the natural progression steps in a process from very coarse to fine-grained inference.

A digital image is most commonly represented by a rectangular cluster of **pixels**. Each pixel gives a discretized and uniform representation of color for the area of the world it covers. Because it discretizes a continuous world, there will be some loss of information, but if the resolution of the image is high enough, this loss can be negligible depending on the application.

2.2 Deep Learning and Neural Networks

Traditional machine-learning methods are limited in how well they process raw data input. They often rely on domain expertise and explicit engineering of features or patterns to extract to make sense of the data. This is also true for image processing methods. *Deep learning* encompasses approaches that structure algorithms in layers into a network. In image processing, this can come in the form of transforming the raw pixel data into information about the presence or absence of edges in different locations at the first layer, then concatenate these edges into more complex features in the later layers. The most important aspect of deep learning and the biggest difference from the traditional methods is that the features emphasized in the processing of data are not engineered by domain-experts, but learned by the method[46].

There are four main types of machine learning:

- *Supervised learning* is the case when an input data point x is provided with a ground truth target t. The methods objective is to find the function f that maps the output $y = f(x)$ such that it matches t.
- *Weakly- or semi-supervised learning* are the supervised methods with noise, incomplete, or otherwise imperfect labels to the data.
- *Unsupervised learning* is the class of methods tasked with finding the underlying structure in the data without any prior information. This usually involves clustering or dimensionality reduction.
- *Reinforcement learning* is the process of letting the method develop a model or policy for the task-environment based on trial and error indicated by rewards received.

This section will provide an introduction to the fundamentals of one of the most used architectures used in deep learning, the neural network.

Artificial Neural Networks are constructs inspired by biological neurons. The artificial neurons that make out these networks are called *perceptrons*, and the networks they make out are sometimes called *multi-layer perceptrons* (fig. 2.1). The perceptron

(fig. 2.2) was developed by Rosenblatt (1961)[62] inspired by previous work done by McCulloch and Pitts (1943)[54]. The perceptrons send a weighted sum of the inputs plus a bias though an *activation function* as an output, as seen in eq. (2.1) The result is then fed into a new layer of perceptrons unless the perceptron is in the output layer. In image processing, the nodes of the input layer usually take in the pixel values.

$$\begin{aligned} z &= \sum w_i x_i + b \\ y &= f(z) \end{aligned} \tag{2.1}$$

The learned parameters in the perceptrons are the *weights* on the connections between the perceptrons in two layers in the network.

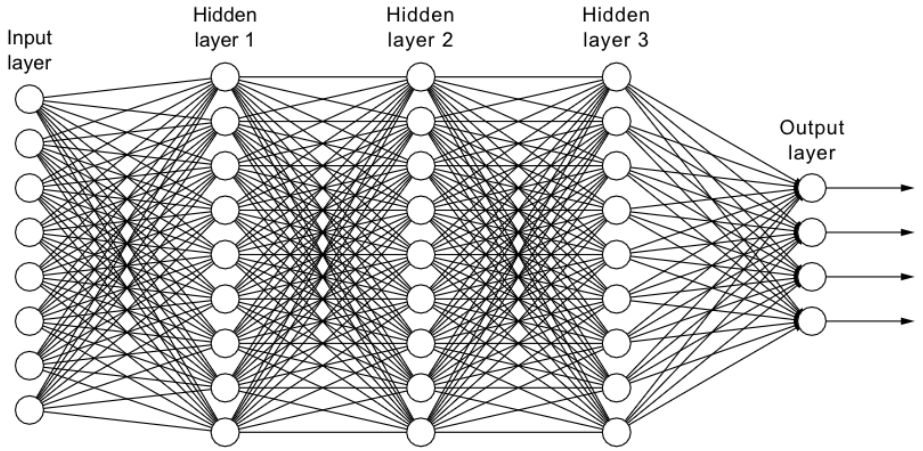


Figure 2.1: Example of a fully connected (fc) neural network.[1]

Training the MLP through supervised learning is the method that has shown the best performance in the field. Properly adjusting the learnable parameters until they properly approximate a map from the input data to their respective target labels is the purpose of the training algorithm.

The loss function also called cost function is usually the measure used to determine

[1]Image from https://freecontent.manning.com/neural-network-architectures/

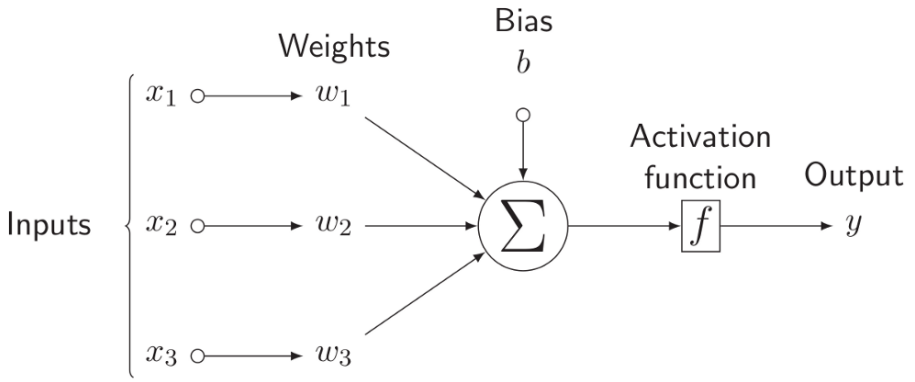


Figure 2.2: Single perceptron.[2]

how closely the function between the input and labels has been approximated. According to *the universal approximation, theorem* as first proved by Cybenko (1989)[21], every continuous function defined in $\mathbb{R}^n$ can be arbitrarily well approximated by a feed-forward artificial neural network with finite neurons in one hidden layer. This theorem is based on an assumption on the activation function, namely that it is non-polynomial[47]. The potential of ANNs does not lie in choosing the optimal activation function, according to Hornik (1991)[39], but the composition of the architecture itself. Equation (2.2) show the definitions of the *mean square error (MSE)* and *binary cross-entropy loss*, which are commonly used loss functions. For a data sample i considered as a class k we denote the ground truth as gt_i ($gt_i = 1$ if sample i is of class k, and $gt_i = 0$ if not) and the predicted probability of sample i being of class k as y_i.

$$\begin{aligned} MSE &= \frac{1}{N}\sum_{i=0}^{N}(gt_i - y_i)^2 \\ BCLE &= -\frac{1}{N}\sum_{i=0}^{N} gt_i \cdot \log(y_i) + (1 - gt_i) \cdot \log(1 - y_i)) \end{aligned} \tag{2.2}$$

[2]Image from https://mc.ai/pytorch-introduction-to-neural-network%E2%80%8A-%E2%80%8Afeedforward-neural-network-model/

There is a multitude of *activation functions* to choose and the following are some of the more popular choices.

- *Logistic/Sigmoid*[33]

$$f(z) = \frac{1}{1 + e^{-z}} \tag{2.3}$$

- *Hyper-tangent*

$$f(z) = \tanh z \tag{2.4}$$

- *Rectified Linear Unit - ReLU*[56]

$$f(z) = \max(0, z) \tag{2.5}$$

- *Leaky ReLU*[52]

$$f(z) = \begin{cases} z, & z \geq 0 \\ \frac{z}{a}, & z < 0 \end{cases} \quad \text{where } a \text{ is a constant.} \tag{2.6}$$

- *Softmax*[17, 57]

$$f(\mathbf{z}) = \begin{bmatrix} f(z_1) \\ \vdots \\ f(z_i) \\ \vdots \\ f(z_n) \end{bmatrix}, \quad f(z_i) = \frac{e^{z_i}}{\sum_{j=0}^{n} e^{z_j}} \tag{2.7}$$

The sigmoid and tanh function were originally popular choices, but later work showed that ReLU and leaky ReLU generally leads to faster convergence [44, 79]. The softmax activation function is usually found in the output layer of a classifier as it normalizes the outputs so they sum to 1.

[3]Image from http://www.programmersought.com/article/1060528072/

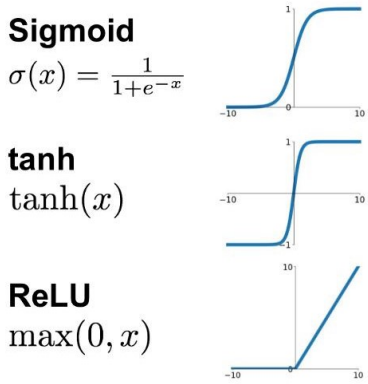


Figure 2.3: Graph representation of the activation functions. Leaky ReLU "leaks" negative values in the negative half-plane.[3]

Epochs and batches are hyper-parameters used in the training process. The data is usually divided into batches for optimal training, and *batch normalization* is the concept of evaluating the loss, performance, or doing adjustment over a whole batch at a time to optimize training speed. One *epoch* is the evaluation of all batches in the set.

Optimizing the loss function is the objective of the training algorithm and the most commonly used method for optimization is *gradient descent* (eq. (2.8)) first proposed by the mathematician Cauchy (1847)[19][15]. This method was further developed and used by Rumelhart et al. (1986)[63] to develop the back-propagation algorithm. Differentiating the loss function with respect to the outputs y_i and applying the chain rule, under certain assumptions to get the gradient with respect to the weights in the previous layer will enable the system to apply the same technique propagating the calculated error backward through the layers. The parameter α is called the *learning rate* and determines how much each training iteration impacts the parameters in the network.

$$\theta_{n+1} = \theta_n - \alpha \nabla L(\theta_n) \tag{2.8}$$

The different hyper-parameters can and should be tweaked to attain optimal results.

2.2.1 Convolutional Neural Networks

The convolutional network was initially conceptualized by Fukushima (1988)[28], but only the architecture was proposed here without any learning algorithm to go with it. Later LeCun et al. (1998)[45] applied the learning algorithm back-propagation[63], and laid the basis of the convolutional network (CNN) used in newer methods in his classification network.

As input to a fully connected network, it doesn't matter how the data is ordered as long as all the data is consistently ordered in the same fixed way. This makes them unable to conserve local contextual connections in the data. The convolutional structure seen in fig. 2.4 makes better use of the local information by the convolution operation. It is a mathematical operation used for filtering images usually for blurring, sharpening, smoothing, edge detection, and more. This is done by convolving a *kernel* matrix or *receptive field* with an image. The definition of a discrete 2-dimensional convolution is shown in eq. (2.9).

$$(g * f)[x, y] = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} f[x, y] g[x - m, y - n] \tag{2.9}$$

Compared to a fully connected layer having a weighted sum going from one layer to another, this is simply a bit more sophisticated operation, and the learned weights can now be found as the entries in the kernels of the different filters in each layer.

The strength of the convolutional architecture is the ability to learn low-level concepts early in the network and higher-level concepts and specialized feature maps later in the network. This is done by aggregating the low-level features by pooling them together. This is one of the fundamental steps in a classification method. Pooling in a CNN is usually done by representing an area in the feature map by either the average or the max value - named average pooling and max pooling respectively. As a network grows deeper it usually grows wider, adding more specialized feature maps

[4]Image from http://what-when-how.com/wp-content/uploads/2012/07/tmp725d63_thumb.png

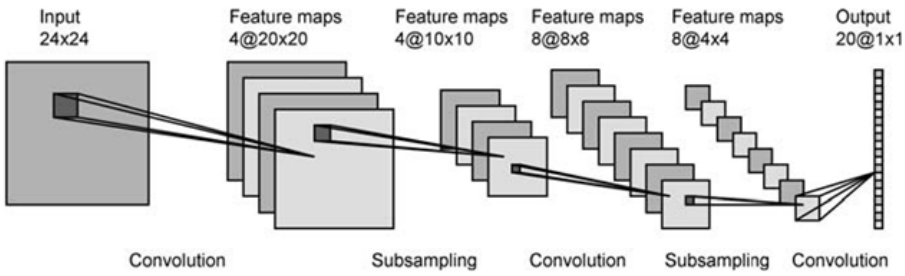


Figure 2.4: An example of a convolutional network structure with increasing number of filters or feature maps per layer and pooling in between to reduce dimentsionality.[4]

to maintain expressiveness. The pooling reduces computational complexity and the added maps increase it.

2.2.2 The evolution of commonly used architectures

Multi-layer feed-forward neural networks are very flexible and can be constructed in virtually an unlimited number of different ways. The performance of the architecture or method applied to a specific problem will depend on aspects including the number of filters per layer, kernel sizes, different types of pooling, optimization, regularization techniques, and activation functions. Having this many different properties to change is what makes these types of networks notorious for being described as "black box" systems. This is also the reason why a lot of the breakthroughs in deep learning with ANN's have come iteratively with new architectures introducing new techniques or a beneficial combination of already known techniques. This section will present some of the most influential architectures. These methods have made such significant contributions to the field that they have become widely accepted standards and their architectures pose as base building blocks for new methods. They were all introduced as winners through the annual ImageNet Large Scale Visual Recognition Challenge[41] (ILSVRC)[5], mainly a classification challenge. The architectures in

[5]http://www.image-net.org/challenges/LSVRC/

question are AlexNet[44], VGG-Net[67], GoogLeNet[70] and ResNet[37] as seen in fig. 2.5.

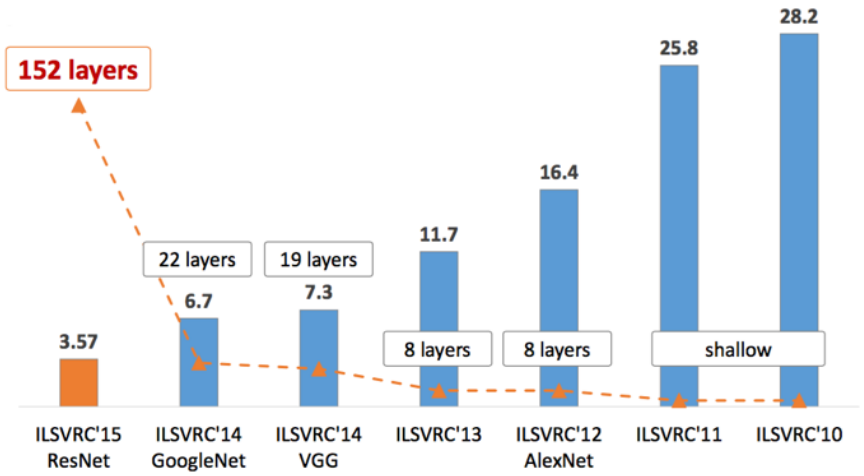


Figure 2.5: Winners of the annual ILSVRC by year.[6] The graphs are showcasing the top-5 classification error-rates achieved by the winning method each year. As a comparison, human error-rate on the same data by an expert annotator was measured to get as low as 5.1% by Russakovsky et al. (2015)[64].

- *AlexNet* proposed by Krizhevsky et al. (2012)[44] was the first deep architecture to win the ILSVRC challenge in 2012. It achieved a top-5 test accuracy of 84.6% compared to the second-best entry using traditional feature engineering methods with an accuracy of 73.8%. This was a huge improvement, solidifying the potential of CNN's in the field. The architecture showed in Figure 2.6, consists of 5 convolutional layers with max-pooling, ReLU activation function, followed by 3 fully connected layers. The convolutional layers produce the downscaled feature vector which is classified by the fully connected layers. It also features dropout to combat overfitting.

[6]Image from https://medium.com/analytics-vidhya/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5

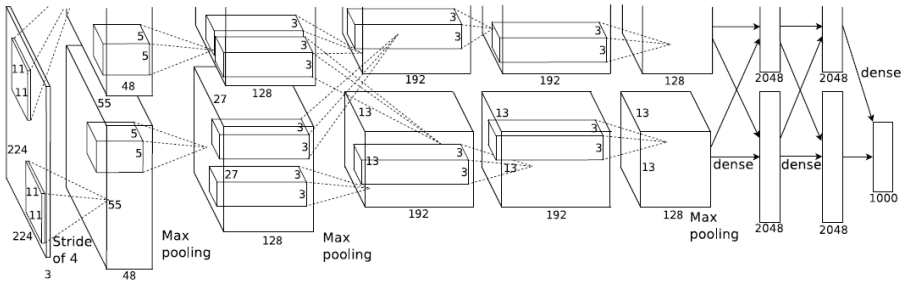


Figure 2.6: Topology of AlexNet, the first CNN to win the ILSVRC. Figure from [44].

- *VGG-Net* proposed by Simonyan and Zisserman (2014)[67] was a set of various models and configurations with slightly different numbers of layers and configurations. The submitted configuration won the ILSVRC-13 challenge and achieved a top-5 test accuracy of 92.7%. This configuration is often referred to as VGG-16 as it had 16 weight layers - 13 convolutional layers and 3 fully connected ones. The main contribution and changes from the previous architectures was more layers making the network deeper, and the use of smaller receptive fields. Where AlexNet used a receptive field of 7×7 in the first convolutional layer then pooling, VGG had three consecutive convolutional layers with receptive fields of 3×3 before pooling. The effective receptive fields in both cases are the same, but the three sequential layers with ReLU activation between each layer results in more non-linearity and almost halving the number of parameters with an equal number of filters. The increased non-linearity through more activations makes the objective function more discriminative making the network easier to train. The reduction in parameters can be seen as a regularization imposed on the effective 7×7 receptive field.

- *GoogLeNet* proposed by Szegedy et al. (2015)[70], winning the ILSVRC-14 challenge followed the trend of being deeper than the previous winners. It consisted of 22 layers but showed a greater complexity than simply stacking layers sequentially. Even though it did not show as significant a leap in performance as the previous years, it showed a top-5 test accuracy of 93.3%. Its structure was

motivated by the fact that a network's performance tends to increase with size. Both depth in terms of the number of layers and with as in the number of filters per layer. A straight forward increase in size by increasing the number of layers and filters increases computational overhead and number of parameters making the networks more prone to over-fitting. Each layer consisted of an inception module (fig. 2.7) performing pooling, large-scaled convolution, and small-scaled convolution in parallel. This network-in-network approach gave a significant gain in quality at a small increase in computational overhead compared to adding the same amount of layers in sequence.

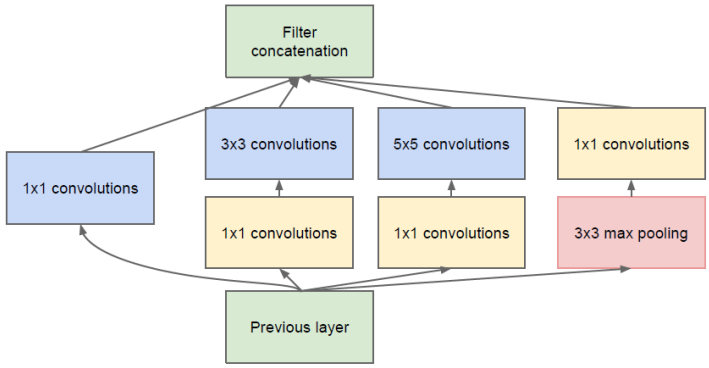


Figure 2.7: Inception building block creating the "network-in-network" structure of GoogLeNet. Figure from [70].

- *ResNet* proposed by He et al. (2016)[37] showed a significant increase in both performance and depth, almost halving the top-5 test error-rate from the previous winner, GoogLeNet. It was the first network to outperform human expert annotators on the test data[64] with a top-5 test accuracy of 96.4%. The best architecture configuration had 152 layers compared to the previous year's winner at 22 layers, and combated the challenges of training such deep a network with the introduction of the residual building block (fig. 3.2). Adding identity skip connections applied to architectures conceptually similar to that of AlexNet[44]

and VGG-19[70] provides later layers with both the output and the unchanged input of previous layers ensuring an emphasis on different features than that of the previous layers. This helps to speed up training and combats the vanishing gradient problem, where the gradient tends toward zero in the back-propagation, increasing the time to or preventing convergence. For more details on this mechanism in network architectures see section 3.3.1.

All these architectures were originally applied to methods of classification. One can argue that this is a task necessary in several other sub tasks of single image processing as well. In the evolution from classification to segmentation, the desired result changes from a single class for the image as a whole to a class for sub-regions of the image. Next, we take a look at how this transition was achieved from the methods mentioned in this section.

2.3 Image Segmentation

Image segmentation is in it's most general form the task of dividing an image into smaller sections or *segments* by grouping pixels together based on some definition of similarity between them. There are multiple different types of segmentation, among them:

- *Semantic* segmentation is classifying each pixel in the image as belonging to a class. This is a more fine-grained inference of determining what class the object in an image is.
- *Instanced* segmentation is semantic segmentation, but now, each pixel is labeled with what object instance of a class it belongs to as well as the class.
- *Panoptic* segmentation is providing additional contextual information to each instance, differentiating between foreground elements and background elements. (or "thing" classes and "stuff" classes as they often are referred to in literature.)

Segmentation of an image can be viewed as a classification task without the as heavy reduction in dimensionality, a more fine-grained inference than simply class

scores for the image as a whole. The naive approach inferred from how a convolutional network works would be to stack convolutional layers without pooling where the last layer would have as many feature maps as there were classes with a softmax activation to have the same resolution on the output as the input. Conserving the resolution of the original image would be very computationally expensive, thus infeasible on larger images. Looking at the success of the classification architectures discussed in section 2.2.2, a similar approach with modifications was proposed by Long et al. (2015)[51]. They were the first to train a fully convolutional network (abbreviated FCN) end-to-end for segmentation. The idea was based on altering the classifiers so they output a classification score for sub-regions of the image instead of the image as a whole. By replacing the fully connected layers in well-established classification methods with convolutional layers (fig. 2.8) the network would output heatmaps of pixels for each class instead of classification scores. These heatmaps were then upscaled by transpose convolutions (also referred to as fractionally strided convolutions or deconvolutions) to make the output of each heatmap the same resolution as the input image for pixel-wise classification. This modification was applied by Long et al. to AlexNet[44], VGG-net[67] and GoogLeNet[70] and showed significant improvement over traditional methods and the previous state-of-the-art method of SDS[35] on the Pascal VOC-{11,12}[27] datasets.

The contributions of Long et al. (2015)[51] with the FCN is considered a cornerstone for segmentation as it showed that convolutional neural networks were capable of efficient learning on arbitrary sized input that beat the state-of-the-art. There are however significant shortcomings on the method inhibiting it from certain applications. Some of the most significant aspects of potential improvement were: inference time as it did not run in real-time, did not consider global contextual information, and had no object instance awareness. As the authors said themselves: "(...) global information resolves *what* while local information resolves *where* (...)". The receptive fields and feature abstraction preserves the local information, but other mechanisms need to be applied for the global information to be considered. The FCN provided a good base as an *encoder-decoder* network for other methods to be developed that would improve on the method's weak points like run-time and instance awareness.

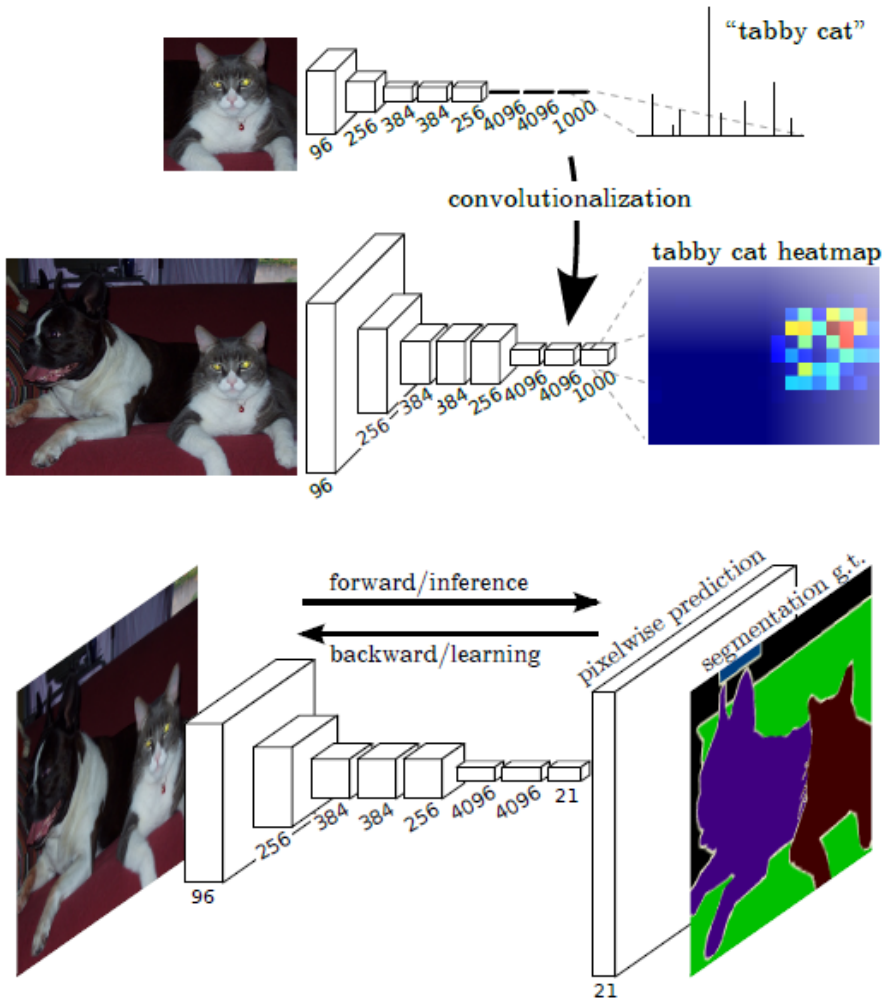


Figure 2.8: Fully convolutional network created from classifier architecture. Figure compounded from [51]

Chapter 3

Methodology and Implementations

This chapter will describe the methods, architectures and implementation frameworks used in this thesis. First, we present Faster[61] and Mask R-CNN[38], two recent methods for object detection and instance segmentation respectively. We detail the contributions of the methods and the mechanisms that makes them perform like they do. Next, we present the architecture models and features compatible with modules of the methods before we give an overview of the implementations software frameworks we are using to run experiments with.

3.1 Faster R-CNN

In this section, we will give an overview of some of the details of the Faster R-CNN algorithm. Faster R-CNN[61] is an iteration in a line of evolving algorithms that are built on the same core concepts, mechanisms, and architecture. It forms the base for Mask R-CNN detailed in section 3.2. The system consists of two modules. The first module contains deep convolutional networks that extract features from an image and

proposes regions of interest (RoI) as input to the detector which is the second module.

Region Proposal Networks

There are many different region proposal methods (for instance [76, 82]) and in effective detection methods [36, 30], generating the region proposals was the test-time bottleneck. With Faster R-CNN Ren et al. proposed the use of a FCN[51] as a *region proposal network*. A region proposal network (RPN) takes an image of any size as input and outputs a set of rectangular region proposals. A sliding window approach is used where multiple region proposals are generated for each window position with the box anchors in the middle of the window.

Multi-task loss

The networks RPN and second module have two separate output layers which output the class prediction and bounding box regression offsets for each RoI from the first module. The classification layer outputs a discrete probability distribution $p = (p_0, \ldots, p_K)$ as the result of softmax activation over $K + 1$ class outputs from a fully connected network. The box layer outputs box offsets $t^k = (t_x^k, t_y^k, t_w^k, t_h^k)$ for each class k given in the parametrization from [61] given in eq. (3.1)[61]. The parametrization is a scale-invariant translation and log-space width and height shift relative to the object proposal. The parameters x, y, w, h denote a box's center coordinates, weight and height and subscript a denote that it's for the anchor box.

$$\begin{aligned} t_x &= \frac{x - x_a}{w_a}, \qquad t_y = \frac{y - y_a}{h_a} \\ t_w &= \log(\frac{w}{w_a}), \quad t_h = \log(\frac{h}{h_a}) \end{aligned} \tag{3.1}$$

The loss function evaluated for each RoI is defined in eq. (3.3)[30] below. The tuple t^u denotes the box predictions for a class u and the tuple $v = (v_x, v_y, v_w, v_h)$ defines the ground truth. The Iverson bracket indicator function $[u \leq 1]$ in eq. (3.3)[30] is evaluated to 1 when $u \leq 1$ and 0 otherwise. This results in a box lsos of 0 for boxes not of the RoI's class.

$$L(p, u, t^u, v) = L_{cls}(p, u) + L_{box}(t^u, v) \tag{3.2}$$

Each of the loss components L_{cls} and L_{box} in eq. (3.2)[30] is defined in eq. (3.3)[30]. The hyper-parameter λ is a weight that controls the balance between the different task losses, but in general it is set to $\lambda = 1$.

$$\begin{aligned} L_{cls}(p, u) &= -\log(p_u) \\ L_{box}(t^u, v) &= \lambda[u \leq 1] L_{loc}(t^u, v) \end{aligned} \tag{3.3}$$

The location offset loss L_{loc} from eq. (3.3)[30] is expressed in eq. (3.4)[30].

$$L_{loc}(t^u, v) = \sum_{i \in \{x,y,w,h\}} \text{smooth}_{L_1}(t_i - v_i) \tag{3.4}$$

The smooth_{L1} from eq. (3.4)[30] is a robust L_1 loss defined in eq. (3.5)[30] that has a low sensitivity to outlier values.

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases} \tag{3.5}$$

For a full image loss seen in eq. (3.6)[61], the average loss contribution from each object prediction from eq. (3.2)[30] is considered.

$$L_{tot} = \frac{1}{N_{cls}} \sum_i L_{cls} + \frac{1}{N_{box}} \sum_i L_{box} \tag{3.6}$$

Shared architecture features

The RPN module and the whole system share convolutional layers in a unified network throughout the first module. The shared layers are trained in an alternating multi-step training routine. First, the RPN is trained and fine-tuned end-to-end for the region proposal task. Next, a separate Fast R-CNN[30] network is trained using the proposals from the RPN. Now the detector and the RPN have separately trained networks and do not share layers. In the third step, the Fast R-CNN layers from the previous step

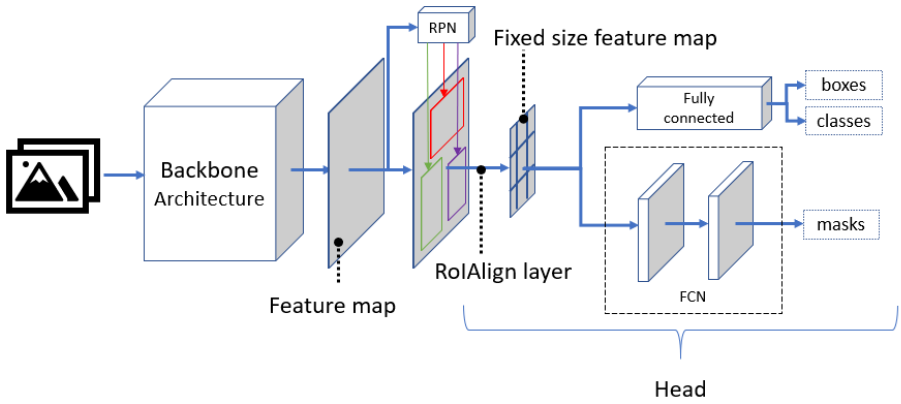


Figure 3.1: The general **Mask R-CNN** architecture.

are used to initialize the base of a new RPN and only the layers unique to the RPN are trained. Lastly, the shared layers and RPN layers are kept fixed while the Fast R-CNN layers are trained. This approach results in the next iteration in the R-CNN family of algorithms, Faster R-CNN named for it's reduced region proposal generation time.

3.2 Mask R-CNN

In this section we elaborate on the key elements of Mask R-CNN. The work by He et al. (2017)[38] mainly adds a branch for segmentation decoupled from the box offset prediction and object classification as well as a mechanism for pixel alignment on RoIs added to it's predecessor Faster R-CNN[61] detailed in section 3.1.

The method is built on the efforts of Ren et al. (2015)[61] and adopts the first stage of Faster R-CNN. The full architecture shown in fig. 3.1 is divided into two stages like it's predecessor, the *backbone* and the *head*. The backbone is identical to that of Faster R-CNN, and it is in the second stage of the algorithm, the head, that the novelty of this method lies.

The most significant addition to the head is a new branch parallel to the branch doing box regression and classification. This branch outputs an $m \times m$ semantic mask

from each RoI by passing the feature map through a fully convolutional network (FCN)[51]. Using an FCN like Long et al. (2015)[51] rather than a fully connected (fc) network like some previous efforts[59, 58, 22], Mask R-CNN achieves more accurate masks prediction with fewer parameters as proved by experiments. Keeping this in a separate branch allows the segmentation network to preserve the spatial layout of the RoI without reducing its dimensionality into a more compact feature representation better suited for classification.

The method is performing more than one task on each RoI which entails considering a multi-task objective function. The total loss is threefold and includes separate losses for the bounding box, class and mask of a RoI, $L = L_{cls} + L_{box} + L_{mask}$. The classification loss L_{cls} and bounding box loss L_{box} are the same as that of the preceding algorithms[30, 61] detailed in section 3.1. The mask loss L_{mask} is defined as the average binary cross-entropy loss (eq. (2.2)) considered over the mask associated with the RoI. The mask branch outputs $K \times m \times m$ dimensional output, which is an $m \times m$ binary mask for each class k. The loss L_{mask} for an RoI classified as class k is only defined on mask k, which means that only the class for the appropriate class is contributing to the loss and competition between classes is avoided in the mask generation. This decouples object classification and mask generation which sets this algorithm apart from the common practice in semantic segmentation using FCNs[51]. It is achieved by considering a *binary* loss over a per-pixel *sigmoid* activation on a per-class basis instead of a *multinomial* loss over a per-pixel *softmax* activation for the mask. This distinction is considered crucial for Mask R-CNN's success over other methods.

RoIAlign is an operation alternative to RoIPool[30] intended to eliminate the misalignment between the RoI and the extracted features from the input image caused by the quantizations applied by the operation. This mechanism is to address the heavy dependency on a high correspondence in the spatial alignment between the pixels in the input image and the feature in the feature map. RoIAlign uses bilinear transformations[36] to calculate the exact values at sampling points in each RoI bin. The results are not sensitive to the number of sampling locations or how many points are sampled as long as no quantization is performed, 4 locations are used for each discrete spatial RoI bin.

3.3 Backbone architectures

This section will give a more in-depth elaboration on the details and structure of the backbone architectures commonly used in visual recognition. This field of research if moving away from "feature engineering" more towards "network engineering"[44, 37, 67, 70]. The difficulty of designing new architectures increases with the number of hyper-parameters as discussed in section 2.2. Some of the networks architectures like discussed in section 2.2.2 like VGG[67] and GoogLeNet[70] has proven themselves to be robust in a wide array of methods[25, 30, 61, 38, 51, 59]. These recent efforts in the field has introduced an emphasis on modularity and a *split-transform-merge* strategy like ResNet[37] blocks from He et al. and the Inception module[70, 71, 72] by Szegedy et al. respectively.

3.3.1 Residual blocks - ResNet & ResNeXt

The modular blocks introduced with ResNet[37] stacked together in different configurations have been used by multiple recent methods [30, 61, 51, 59]. They also form the basis for all the different architecture modules used in Mask R-CNN[38] in the implementations presented in section 3.4.2. In table 3.1 the baseline structural configurations available in that framework are presented.

The residual blocks with skip connections feature multiple benefits[37] without increasing structural or parametric complexity from the equivalent plain structure. Some of these benefits are that they handle identity mappings in deeper structures without driving the weights to zero and easier optimization because of the "shortcuts" introduced between layers. It also allows for the training of deeper networks as the identity input from earlier layers ensure that layers further down learn something else increasing accuracy.

Taking inspiration from the inception module's[70, 71, 72] *split-transform-merge* tactic the modified residual blocks of ResNeXt were designed with internal parallel paths by Xie et al. (2017)[78]. Their results show that increasing the *cardinality* (number of parallel paths) in a block is a more effective way of gaining accuracy than increasing the width (number of filters) or depth (number of layers) of the network. Figure

stage	output	ResNet-50	ResNet-101	**ResNeXt-101**
conv1	112×112	7×7, 64, stride 2		
conv2	56×56	3×3 max pool, stride 2		
		$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64, * \\ 1\times1, 256 \end{bmatrix} \times 3$
conv3	28×28	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128, * \\ 1\times1, 512 \end{bmatrix} \times 4$
conv4	14×14	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256, * \\ 1\times1, 1024 \end{bmatrix} \times 23$
conv5	7×7	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512, * \\ 1\times1, 2048 \end{bmatrix} \times 3$
	1×1	global average pool, 1000-d fc, softmax		
# params.		$\mathbf{25.5} \times 10^6$	$\mathbf{44.0} \times 10^6$	
FLOPs		$\mathbf{3.8}$ - $\mathbf{4.1} \times 10^9$ **	$\mathbf{7.8} \times 10^9$	

Table 3.1: The architectural structure of the ResNet[37] and ResNeXt[78] networks at different depths. Inside the brackets we find the shapes of a residual block and outside the brackets the number of blocks stacked in the given stage. FLOPs means Floating-point Operations Per second.
* Cardinality $C = 32$, eg. grouped convolutions with 32 groups.
***[37, 78] disagree on this value*

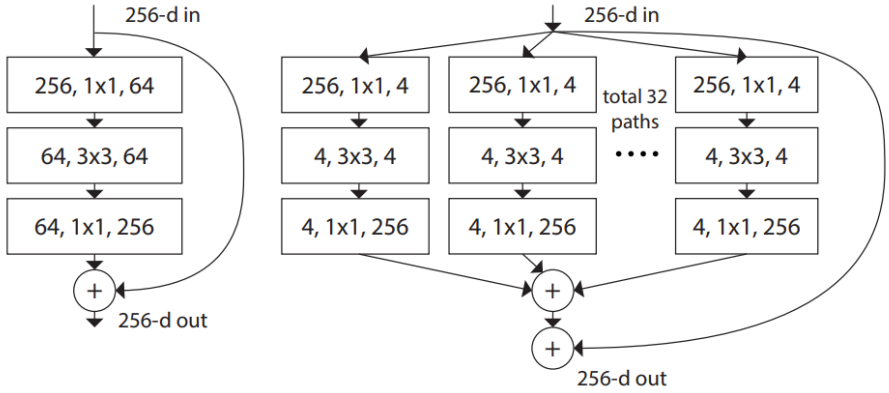


Figure 3.2: **Left:** The basic residual block module of ResNet[37] with skip connection. **Right:** A block of ResNeXt[78], with $C = 32$. Each layer is represented as [# channels in, filter size, # channels out]. This figure is from[78].

3.2 illustrates this evolution to the block modules of the ResNeXt[78]. The blocks are subject to two rules so a *block template* could be designed so all modules can be determined according to the same rules. These are (i) blocks producing spatial maps of the same size have the same hyperparameters (width and filter size) and (ii) the downsampling of the spatial map is inverse proportional to the number of filters to keep computational complexity.

Because the concept of Mask R-CNN is a very general and highly modulated algorithm the modules can contain different network sub-architectures as long as the input and output dimensions match. In fig. 3.3 two different examples of configurations for the architecture head are depicted.

3.3.2 Feature Pyramid Networks

Learning features and detecting objects on multiple scales in recognition systems either requires multi-scale training or other mechanisms to introduce scale-invariance to the performance of a convolutional network. Feature pyramids are basic components in mechanisms to handle this challenge.

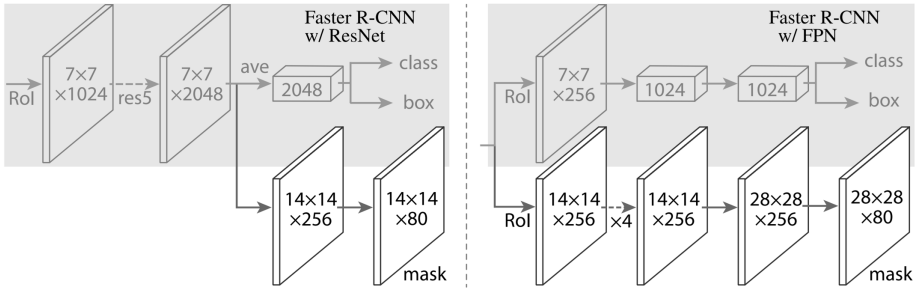


Figure 3.3: Two different configurations of the head in Mask R-CNN. The Faster R-CNN[61] head is extended by the ResNet[37] C4 (Left) and FPN[49] (Right) backbones. This figure is from [38].

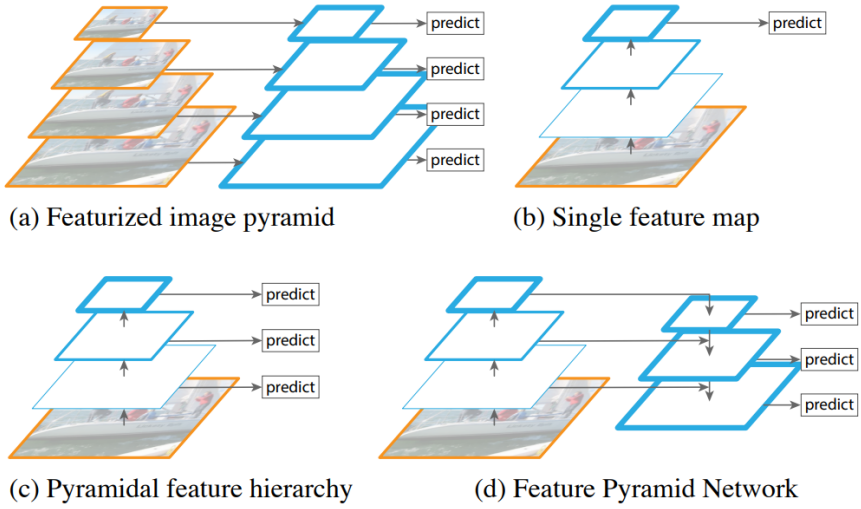


Figure 3.4: The topological layout of multiple different alternatives for feature maps in a pyramid structure. This figure is from [49].

Recent detectors like Faster R-CNN[61] have avoided using pyramid representations because they are expensive in terms of computation time and memory. Lin et al. (2017)[49] propose a multi-scale pyramidal hierarchy of deep CNNs with marginal extra cost. In fig. 3.4 we see different alternatives of pyramid structures. Alternative (a) is slow because it computes feature maps on different image scales independently, (b) uses only single scale detection for faster inference, but this limits the performance and range of detectable objects and in (c) the feature hierarchies in a CNN are reused as if it already was a featured pyramid structure. Combining multiple of these approaches into (d) the inherent FPN from [49], we keep the speed of (b) and (c) while also increasing accuracy like (a).

Using the feature activation from the last residual block in each of the different convolutional stages past the first one in the ResNet[37] architectures supports the desired pyramidal structure. Adding skip connections between the feature maps of different scales for bottom-up and top-down pathways between the differently scaled feature maps enhances the representations in the other layers.

The new structure of the FPN[49] provides a powerful at a marginally increased cost to already existing recognition frameworks or for the use in modular building blocks like RPNs(section 3.1).

3.4 Implementation frameworks

The modular and flexible nature of deep learning algorithms opens the possibilities of implementing them in virtually infinite different ways in software systems. In this section we present software libraries providing building block tools and fully implemented software systems used for the experiments conducted in this thesis.

3.4.1 Various libraries

Torch[20] is an open-source machine learning software library made available to the scientific community as a tool to simplify the comparison, extension, and even addition to learning algorithms. The core package of torch provides a flexible N-dimensional

array, a tensor, which supports a multitude of manipulative and mathematical operations. The library also features a modular way to instantiate a framework for neural network architectures (section 2.2). This provides an easy way to build a network, defining its parameters and interconnections between layers as well as forward and backward passes to automate inference of data samples and backpropagation while training. As of 2018, Torch is no longer in development[8].

Caffe[42] (Convolutional Architecture for Fast Feature Embedding) is an open-source machine learning software library similar to Torch. It is implemented in C++ and features bindings to Python and MATLAB. It is mainly developed at UC Berkley[1], but has a lot of contributors today.

Caffe2 is yet another library originating at Facebook. Today it is a deprecated project and it was merged into PyTorch in 2018.

PyTorch is a Python package based on the torch library. It is the last iteration in the Torch/Caffe family of libraries and features a dynamic definition of computational graphs in contrast to other libraries like TensorFlow which requires the entire graph to be defined before you can run your models.

3.4.2 Detectron2

Detectron2 is Facebook AI Research's (FAIR) software system that implements state-of-the-art object detection algorithms. It is the finished implementation of maskrcnn-benchmark[53], a ground-up rewrite in PyTorch of its predecessor Detectron[31] which was based on Caffe2. The purpose of these projects is to further develop upon the flexibility and support provided by their building block learning frameworks to facilitate research for object detection. In the systems, there are multiple implementations of popular algorithms built on several different backbone model architectures in addition to the possibility of rapid implementation and evaluation of novel algorithms and architectures.

In the Detectron system the following algorithms are implemented and available:

- R-FCN[23]
- Fast R-CNN[30]

- Faster R-CNN[61]
- RetinaNet[50]
- Mask R-CNN[38]

The following backbone architectures are available:

- VGG16[67]
- Feature Pyramid Networks (FPN)[49] (with the below arhitectures)
- ResNet[37], both the 50, 101 and 152 layer versions
- ResNeXt[78], both the 50, 101 and 152 layer versions

In this thesis, we only elaborate on the algorithms and backbone architectures used or evaluated. For details about the other implementations available in Detectron2, see the referenced sources.

3.4.3 The PySilCam software suite

The PySilCam[60] suite developed by Davies et al. (2017)[24] from SINTEF is a pipeline used for in-situ image processing on a lightweight autonomous underwater vehicle (LAUV). The system features an image processing pipeline suited for real-time sequential image segmentation and object extraction for classification. The images are corrected by a clean background to reduce noise, then a segmentation mask is generated for the images by traditional clustering methods based on binary thresholding. Based on the clustering of thresholded pixels in the segmentation mask object regions are defined and particles are extracted based on the segmentation areas. After extraction, the image segments are classified. At the end of a processing sequence, the data from all the extracted objects are saved and made available for post-processing.

This software suite used in-situ has provided all the custom imaging data used in this thesis.

Chapter 4

Datasets

In this section, we will discuss the importance of data quality and present the data used in experiments conducted for this thesis.

Data is arguably one of the most important factors in a machine learning task. What kind of data is available will heavily influence what method or approach should be chosen and how the said approach will perform. Compiling the necessary quantity of data of sufficient quality and representing it in with relevant information to aid the learning method can easily be the most challenging part of a machine-learning task. The dataset needs to have a large enough scale to capture the full scale of the problem and needs to be structured in a way that is efficient for the system to analyze. This is especially true for deep learning architectures. The effort of constructing such datasets is extremely time consuming and requires a lot of resources. Manually annotating an image with near-perfect pixel-wise labeling with all relevant information usable in fully supervised learning can take up to several hours per image. Because this is such a critical part across all tasks in the field of machine learning and by consequence image segmentation a wide arrange of standardized datasets have been constructed by the research community for easy comparison between different systems.

4.1 Microsoft Common Objects in Context (COCO)

MS COCO[48] is a recent large scale dataset for holistic scene understanding associated with segmentation and captioning. It is used as a dataset in several challenges with detection being the most relevant ones for this setting. That particular dataset consists of more than 80 classes, 200 000 images divided roughly 40% - 20% - 40% into train, validation, and test images. The test set is further divided into subsets for extra validation of challenge results. The results of this challenge are presented at the European Conference on Computer Vision (ECCV)[1] annually together with that of ILSVRC. It has grown in popularity over the past years due to its large scale, and the creators arguing that it can train object detectors with better localization capabilities than other datasets.[29, 81]

4.2 Custom Planktonic Dataset - Copepod-petridish

There are datasets containing millions of microscopic images of planktonic organisms made available from several different research groups around the world. An overview was provided in Bergum (2019)[13], but is omitted here as it is not relevant for the scope of this thesis. The WHOI[69] dataset is one of these datasets with high quality. The data consists of single organism image segments for classification with complete annotation labels. This data is not suited for our task however as we seek to extract the single object regions from full image scenes. You can say that the data available in [69] matches the output but not the input of the process we seek to develop.

The emphasis on supervised learning in this project founded a need for available data annotated for segmentation or object detection. To the best of our knowledge, there are no published datasets for this purpose available anywhere. This made it clear that producing such a dataset was a necessary contribution in order to conduct meaningful experiments. In this section we will go over the details of the novel data, aspects of the labeling process, and the resulting dataset referred to as **ailaron-copepod-petridish**[14].

[1] http://image-net.org/challenges/ilsvrc+coco2016

The images

The images used to build the custom dataset was captured by the SilCam in a lab environment. In this experiment, one of the types of planktonic organisms of interest for this project, copepods, were passed in front of the camera to guarantee the capture of the organisms. The collection consists of 131 images of resolution 2448×2050. This set of data was chosen because of the quantity, quality, and properties of the samples. There are relatively few images making it possible to verify that they are of decent quality without major noise or distortions. These images were already processed by the PySilCam suite and readily available with object exports, binary segmentation masks, inferred stats, and background-corrected images as outputs from the PySilCam Software Suite.

Challenges connected to individual data samples: Many factors are playing into what quality the data available has. This includes what the data is representing, what equipment is used to sample it, and the nature of the environment it is sampled in. Below are some phenomenons that typically can cause problems for not only segmentation algorithms, but possibly other image processing tasks as well. They are very problem dependent and will potentially affect different models to a varying degree[73]. The first ones are usually subject to the conditions of the equipment or environment while the latter are usually subject to what the data is representing.

- *Blur* can occur due to multiple reasons. It can be due to problems in the camera model or lens adjustment or focus, rapid movement, or disturbing elements like for example smoke. This effect will make contours between regions blend into one another or be otherwise misrepresented. This effect does not show a dominating presence in the raw images but has been reported to have a greater effect on the background corrected samples.

- *Vignetting* is the effect of an image having a border around the edges and corners that is or appears darker than the rest. This can occur because of filters or the structure of the lens casing blocking light in certain areas. It is a common occurrence in unprocessed microscopic images. The image scene has a great

variation in illumination, vignetting occurring towards the edges except for the top edge of the image.

- *Occlusion* is when something is fully or partially hidden by something else. This can happen due to the viewpoint of the camera or objects relative position to one another. A single object can also perform *self-occlusion* by attaining different poses making some parts of the object occlude other parts. Only parts of an object being visible can affect an algorithm's ability to properly detect it. Self-occlusion is a common occurrence with one of the classes obscuring some of the characteristic features of the objects.

- *Transparency* is a form of occlusion creating a problem of definition. If one object is seen through another transparent object, which class does the pixel(s) in question belong to, the object in the background of the transparent object? The object seen through the transparent object can also become warped, potentially lowering performance. Most object instances in the data show a degree of transparency, but very few are overlapping so we see one though the other. The transparency issue will rather make it harder for the method to distinguish the objects from the background.

Image Annotation

Image annotation is the task of annotating the contents of an image with labels. What kind of annotations an image is provided depends on the task the image is used for and the agent annotating the labels. Unless otherwise expressed, image annotation is in this part referring to the human-powered task of manual annotation. Efforts to develop frameworks to automatically annotate images, such as [10, 12], have been made, but these frameworks also require a baseline of manually annotated data to train on. Most of these types of frameworks however are dependent on a baseline of manually annotated images to develop them. There are a lot of aspects to consider when determining what kind of information and what level of detail on that information one

applies to the annotations. An elaboration on methods and aspects of image annotation can be found in the work of Hanbury (2008)[34].

To create annotated images three things are needed:

- Images
- An person to annotate the images
- A tool or platform to annotate the images on

The images used are those captured from the system mentioned in section 3.4.3. The person annotating the images is the author of this thesis. The platform used to annotate the images has a wider range of alternatives than the two previous points. Several tools and platforms were considered such as LabelImg[75], TrainingData.io[74] and LabelMe[65]. More examples can be found in an overview by Morikawa (2019)[55]. Among these the VGG image annotator tool[26] is used to label the images in this project. This tool was chosen over the others based on how available it is to use, the features it provides and the effort needed to invest in making it run and learn how to use it.

VGG Image Annotation Tool (VIA)

The tool is a simple and standalone software kit suited for the annotation of both audio, images, and video. The tool runs in a browser and requires no setup or installation before determining the settings of the annotations. It is an open-source project based on HTML, Javascript, and CSS without any dependency on external libraries. It is developed by the Visual Geometry Group (VGG) from the University of Oxford. It is licensed as BSD-2 which allows the use in both commercial applications and academic projects, such as this one.

The labeling process

Manually annotating images is a strenuous task. The result will depend on several different factors. The amount of information included in the annotation, the precision

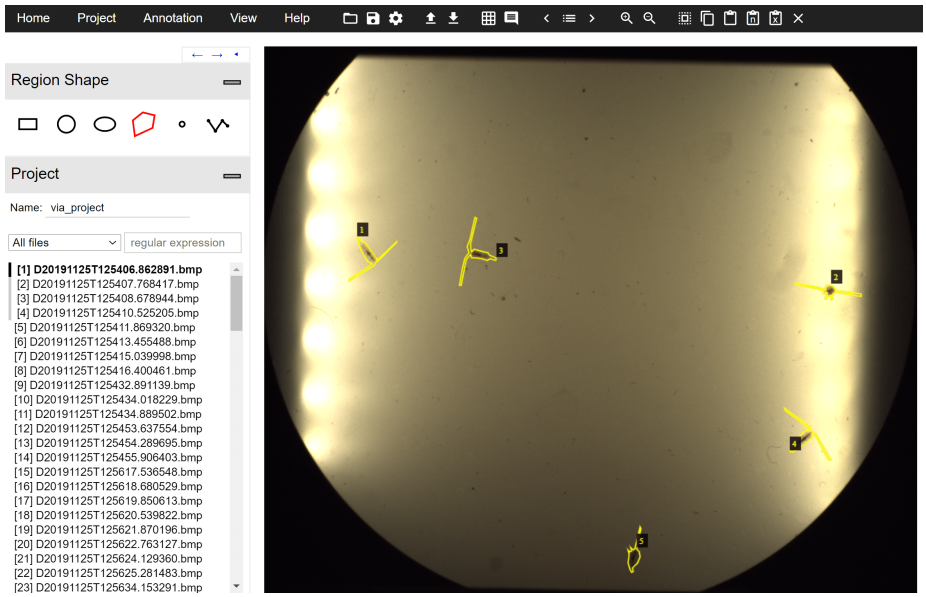


Figure 4.1: Image scene in the VGG Image annotation tool.

ID	Description
blur	The object or region around the object is blurred.
light	The object is in a part of the image that is more illuminated than other areas.
vignetting	The object is located towards the border of the image in a darker part cause by vignetting.
self_occlusion	The object is oriented in such a way that it is occluding large parts of itself. This is also used in the extremely rare case that it is occluded by another object.
edge	The object is partly outside the image.

Table 4.1: The ID's and descriptions of the region properties used in the VIA from fig. 4.2b during the labeling process.

of the labeling, and the tools used are some factors that can have an impact on how well the annotations turn out. Some degree of inaccuracy and error must be assumed to be associated with the annotations applied to the data. This is a natural consequence of the limited resources available in this type of project.

The most obvious property to apply to a region is the category of the object in the region. This metadata is defined as `thing_classes = ['oil', 'other', 'bubble', 'faecal_pellets', 'copepod', 'diatom_chain', 'oily_gas']` as in fig. 4.2a in order to keep the same format as is output from the PySilCam for continuity and format unity. It is important to note that the categories are 1-indexed in the COCO format, but 0-indexed in the standard Detectron2 format.

Because the task of annotation preferably should only have to be done once over a set of data, additional information that is considered valuable if applied in the right way was included. Some of these annotations are redundant in the scope of this project as they are not put to use in the experiments conducted here, but the information was included in the files published together with the ailaron-copepod-petridish dataset. The properties in question can be seen in table 4.1.

It is important to note that even though there are 7 classes included in the description and properties here, only 1 class is used in this dataset. All annotated objects are

of type "copepod". The extra classes and indices are included for the sake of unity to the output of the PySilCam during the image processing.

Set division

To achieve unbiased evaluation results, it is good practice to divide a dataset into different subsets for different purposes. A common way to divide a set is into approximately 70% - 15% - 15% of *training*, *validation* and *test* samples respectively. One example of how to use the different sets can be to use the training samples to update weights during training, the validation set to tune hyperparameters, and the test set to benchmark performance. We have divided our dataset as follows:

- 88 training samples with 541 instances of copepods
- 19 validation images with 115 instances of copepods
- 19 test images with 120 instances of copepods

The split is intended to create separate samples for different purposes in the process of training a model. The training samples should be used to update weight during training while evaluating the validation set at a certain integration interval to tune hyper-parameters. When a model is fully trained its performance can be benchmarked over the test samples. This separation of samples ensure good practice in terms of objective evaluation metrics and avoid under- or overfitting the model.

Data formats

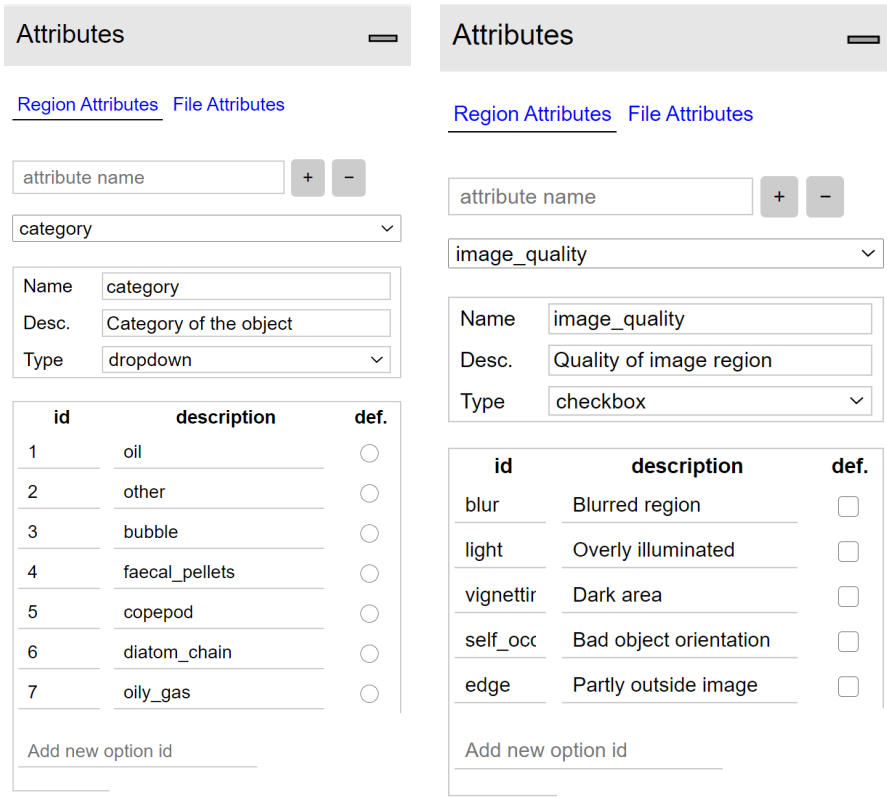
The ailaron-copepod-petridish is made available both in the standard MS COCO data format and in the standard Detectron2 data format at [14]. In the conversion, the raw files exported from the VIA tool to the COCO format and the standard Detectron2 format the information on the region properties are lost as there seemed to be no appropriate data fields for this information.

The Detectron2 standard dataset dictionaries contain both required and optional fields. The dataset format is in the form of a `list[dict]` with specifications similiar

to COCO's json format. Each dictionary element in the list contains the information about one image. The data fields used are the following:

- `"file_name"`: string, the full directory path to the image file.
- `"height", "width"`: integers, the pixel shapes of the image.
- `"annotations"`: list[dict], each dictionary contains the annotations for one object instance in the image
  - `"bbox"`: list[float] 4 numbers representing the bounding box coordinates.
  - `"bbox_mode"`: structures.BoxMode, the modes are `XYXY_ABS` and `XYHW_ABS`, representing either `[xmin,ymin,xmax,ymax]` or `[xmin,ymin,height,width]` respectively. The BoxMode type is not JSON serializable by default and is not included in the json files. `XYXY` has mostly been used, but `XYHW` is the standard Coco format.
  - `"category_id"`: int, in the range of [0,num_classes-1].
  - `"segmentation"`: list[list[float]], represents a list of polygons. Each list[float] is one simply connected polygon.

For a full overview of available data fields for the standard dataset dictionary format used by Detectron2 see [6].



(a) (b)

Figure 4.2: These are the property options used in the VIA

Chapter 5

Evaluation Metrics

To measure how well a learned method is performing its task, we need some objective metrics of evaluation. These should be standard and well known to reflect the contribution of a new method compared to other systems. Depending on the task and different properties of the system, some aspects of the performance of the method may be of greater importance or significance than others and if they are mutually exclusive, a compromise must be met. This usually applies to the aspect of accuracy versus overhead in the form of inference time, memory usage, and training time.

5.1 Memory and run time

Memory and run-time are often regarded as less important than other metrics because they will vary wildly depending on hardware and implementational details. Yielding some indication or qualitative assessment of the overhead conditioned to achieve similar results is often helpful to the community for other researchers to evaluate whether a system will be suitable to their system or not. Single image scene labeling and real-time video analysis, for instance, have quite different requirements on inference time. Some specialized embedded systems might not have the computational power or memory capacity recommended or required for certain methods intended for GPU

accelerated deep networks. In these cases, trade-offs[40] between speed and accuracy might be necessary. Because the frameworks explored in this thesis are intended for in-situ analysis, reporting, and considering the computational overhead on inference is important.

5.2 Accuracy

Accuracy is most commonly used as a comparison when proposing new methods as this usually is an indication of the theoretical potential achieved in the approach used. There several different ways to measure accuracy, later in this chapter we will briefly describe some of the most popular ones.

We assume to have $K + 1$ different classes labeled as $C_0, C_1, \ldots, C_K$ where C_0 often is defined as the void class indicating no class or background.

- A *true positive*(TP) occurs when the sample is predicted to belong to class C_k, and the ground truth is C_k.
- A *false positive*(FP) occurs when the sample is predicted to belong to class C_k, but the ground truth is not C_k.
- A *false negative*(FN) occurs when the sample is predicted to belong to a class other than C_k, but the ground truth is C_k.

For a single pixel, this is obvious, but for the detection of an object, this is generally determined whenever the *IoU* between the prediction and ground truth is greater than some threshold. In the following p_{ij} is the amount of pixels of class i inferred to belong to class j. This means that p_{ii} are true positive pixels, p_{ij} are false positives and p_{ji} are false negatives.

5.3 Intersection over union

Intersection over Union (IoU) measures the overlap between two boundaries. In segmentation we operate with two different types of boundaries, bounding boxes, and

segmentation masks. It can be defined as the number of pixels in common between ground truth and the prediction divided by the total number of pixels in the combined area. The mean IoU is the IoU scores averaged on a per-class basis. This is usually the standard metric for semantic segmentation purposes.

$$\begin{aligned} IoU &= \sum_{i=0}^{k} \frac{p_{ii}}{\sum_{j=0}^{k} p_{ij} + \sum_{j=0}^{k} p_{ji} - p_{ii}} \\ mIoU &= \frac{1}{k+1} \sum_{i=0}^{k} \frac{p_{ii}}{\sum_{j=0}^{k} p_{ij} + \sum_{j=0}^{k} p_{ji} - p_{ii}} \end{aligned} \tag{5.1}$$

5.4 Precision and Recall

Precision is a measure of how accurate your predictions are, this means how many of your predictions are correct. *Recall* is a measure of how many of the correct predictions you made.

$$\begin{aligned} Precision &= \frac{TP}{TP + FP} \\ Recall &= \frac{TP}{TP + FN} \end{aligned} \tag{5.2}$$

Observing the relationship between precision and recall can yield another useful metric, the *Average Precision*[7]. It represents a weighted mean of precisions with the increase in recall from a previous ratio threshold as the weight. In eq. (5.3) the mathematical definition of AP is defined where R_n and P_n is the Recall and precision respectively at the nth threshold.

$$AP = \sum_{n} (R_n - R_{n-1}) P_n \tag{5.3}$$

5.5 Metrics used in the experiments

This is an overview and explanation of the specific metrics we will be presenting and comparing in chapter 6. These are some of the metrics available in the evaluator

module suited for COCO evaluation[2] in the Detectron2 framework.

AP is an abbreviation for *average precision* and unless specified otherwise it is averaged over multiple different IoU values. This is normally called the mean average precision (mAP). Specifically for the COCO metric, a range of [0.50:0.05:0.95] (10 different values) is used in the Detectron2 implementation. This is different from the most commonly used traditional AP, which is computed at a fixed IoU value of 0.50.

- $\mathbf{AP}_{50}$ is the traditional AP computed at an IoU value of 0.50. This is the standard metric used in some settings[27].
- $\mathbf{AP}_{75}$ is the AP computed at an IoU value of 0.75 and is considered a fairly strict metric.
- $\mathbf{AP}_S$ is the AP for small objects with an area < 32^2 pixels.
- $\mathbf{AP}_M$ is the AP for medium objects with an 32^2 < area < 96^2 pixels.
- $\mathbf{AP}_L$ is the AP for large objects with an area > 96^2 pixels.

The IoU percentage thresholds can be misleading as they are not equivalent to *overlap*. An IoU of 0.5 is equivalent to $\frac{2}{3}$ of a boundary overlapping with $\frac{2}{3}$ of another boundary. To reach 0.75 IoU almost 86% of the boundaries must overlap.

These metrics are considered over both the bounding boxes and the segmentation masks of the objects, distinguished using a superscript bb (AP^{bb}) for bounding boxes and m for masks (AP^m).

Training time is presented in terms of the total amount of seconds taken per training iteration (s/it) and the **inference time** is presented in terms of the total amount of seconds to infer predictions on an image (s/im).

Chapter 6

Results

In this chapter, the findings and results from experiments are stated. For an elaboration on the different metrics, how they are evaluated, and what they entail see chapter 5. First, we state and validate the results presented by the authors of the methods used in [38], then we present the results from our novel experiments. In the end, we summarize and compare the results to verify that they are reasonable, relevant, and valid. All results presented of a particular model are *single-model* results, meaning they are all procured from the same model and not the best values selected from multiple different models.

The notation of the model shorthand follows the format of [BackboneType]-[depth]-[feature]-[learning schedule]. For all the models used in the experiments, we initialize the weights to a state resulting from training over the Coco dataset. The learning schedule is how much training has been done by the model we import the weights from. Most of the models are trained with a 3x schedule which is equivalent to approximately 37 COCO epochs[1], while some only have 1x which is approximately 12 COCO epochs. For example, R50-C4-1x means a stage 4 ResNet-50 backbone on the 1x training schedule, while X101-FPN-3x means an FPN head architecture and a ResNeXt-101

[1]This is training for one epoch over the Coco dataset. Explanations on the coco dataset and what an epoch is can be found in section 4.1 and epoch in section 2.2

Backbone	AP^{bb}	AP^{bb}_{50}	AP^{bb}_{75}	AP^{bb}_{S}	AP^{bb}_{M}	AP^{bb}_{L}
R101-FPN	38.2	60.3	41.7	20.1	41.1	50.2
X101-FPN	39.8	62.3	43.4	22.1	43.2	51.2

Table 6.1: **Object Detection** *bounding box* AP on COCO `test-dev`[48] of Mask R-CNN. See table 6.6 for comparison to the custom dataset. This is a table recreated from parts of **table 3** in [38]. This reference should be examined further for more detailed metrics.

backbone on the 3x schedule. For more details on the differences in architecture see chapter 3. We use the same hyperparameters in training for all the models. We train for 20 000 iterations with a learning rate warm-up schedule[2]to a base learning rate of 0.00025.. We use a training batch size of 5 and the number of region proposals offered by the RPN is the default values for each of the backbone architectures[4]. Most of the models seem to have stabilized without overfitting at this number of iterations (see fig. 6.1)

For an explanation of the different evaluation metrics stated in this chapter, see chapter 5.

6.1 Results on existing top-quality datasets

Tables 6.1 and 6.2 show the results from He et al. (2017)[38] running the Mask R-CNN algorithm over the Coco dataset used in their paper presenting the method. Using the best architecture, ResNeXt101-101-FPN, Mask R-CNN further improves results over the single-model results they present from other models. They show an increase in single-task performance from the benefits of multi-task training and new aligning mechanism which separates the method from Faster R-CNN.

Table 6.3 shows the results when evaluating some of the trained Mask R-CNN models available in the Detectron framework. We show the AP metrics for object detection and instance segmentation masks over the 2017 minival set consisting of

[2]See [4] for details.

Backbone	AP^m	AP^m_{50}	AP^m_{75}	AP^m_S	AP^m_M	AP^m_L
R101-C4	33.1	54.9	34.8	12.1	35.6	51.1
R101-FPN	35.7	58.8	37.8	15.5	38.1	52.4
X101-FPN	37.1	60.0	39.4	16.9	39.9	53.5

Table 6.2: **Instance segmentation** *mask* AP on COCO `test-dev`[48] of Mask R-CNN. See table 6.7 for comparison to the custom dataset. This is a table recreated from parts of **table 1** in [38]. This reference should be examined further for more detailed metrics.

Backbone	AP^{bb}	AP^{bb}_{50}	AP^{bb}_{75}	AP^m	AP^m_{50}	AP^m_{75}
R101-C4-3x	42.576	62.121	46.048	36.652	58.478	39.256
R101-FPN-3x	42.928	63.323	46.834	38.629	60.449	41.271
X101-FPN-32-8d-3x	44.275	64.463	48.618	39.520	61.696	42.570

Table 6.3: **Instance segmentation** *mask* AP and **Instance segmentation** *bbox* AP on `coco-minival-2017`[48] of Mask R-CNN. See table 6.1 and table 6.2 for comparison to the paper results. These are metrics from experiments trying to recreate the results of [38].

Backbone	AP^{bb}	AP^{bb}_{50}	AP^{bb}_{75}	AP^{bb}_{S}	AP^{bb}_{M}	AP^{bb}_{L}
R101-C4-3x	65.231	97.283	78.054	-	51.153	66.320
R101-FPN-3x	64.772	95.453	76.115	-	52.173	65.700
X101-FPN-32x8d-3x	67.736	94.155	80.087	-	62.633	68.337

Table 6.4: **Object Detection** *bounding box* APs over `ailaron-copepod-petridish-test` from some select backbones with available checkpoints for Faster R-CNN in Detectron2 framework.

5000 iamges. For a full per-category rundown see appendix C.

6.2 Results on the Custom Planktonic Data

In this section, we present the results of the experiments on the ailaron-copepod-petridish[14] data using our training procedure on multiple different architectures.

Table 6.4 shows results from Faster R-CNN implementation in detectron2. He et al. (2017)[38] shows an increase in the individual performances of the bounding box and segmentation mask accuracy by considering a loss over multiple tasks in the training process. Our results do not show the same performance increase, but rather very similar results to that of table 6.6.

Our ailaron-copepod-petridish is represented in two different formats, the Detectron standard dictionary format, and the Coco format (see chapter 4). Because we are using the built-in Coco Evaluator[5] we consistently state the results from evaluating over the coco format datasets unless otherwise stated as the standard Detectron format is automatically changed to the coco format by the implementation. The manual conversion that is done by custom code (appendix B) and the automatic conversion do not match, however. In table 6.5 some results when evaluating over the Detectron format dataset after automatic conversion can be found. These are included to illustrate the differences from other results (table 6.6 & table 6.7). The values do not differ significantly from the perspective of the AP values. The object sizes are somehow calculated differently, with no significant impact on the AP.

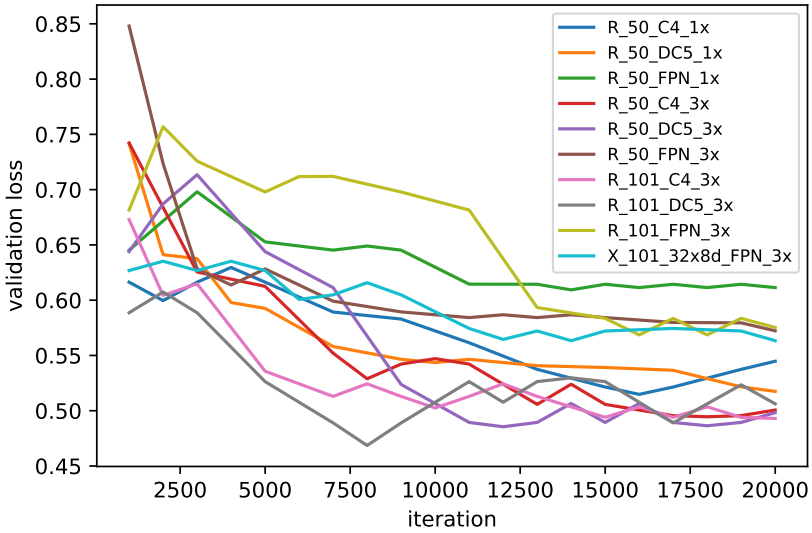


Figure 6.1: **Validation loss** on `ailaron-copepod-petridish-val` every evaluation step during training.

Backbone	AP^{bb}	AP^{bb}_{50}	AP^{bb}_{75}	AP^{bb}_{M}	AP^{m}	AP^{m}_{50}	AP^{m}_{75}	AP^{m}_{M}
R50-C4-3x	66.461	99.101	78.683	71.094	17.486	89.281	0.083	66.461
R50-DC5-3x	61.328	94.167	74.064	67.731	25.955	93.337	0.459	25.957
R50-FPN-3x	65.356	93.766	80.526	71.245	39.161	95.919	3.542	39.172
X101-FPN-3x	67.789	96.513	80.774	72.944	42.020	99.715	7.628	42.033

Table 6.5: Some sample results on evaluation over the detectron2 format datasets.

Backbone	AP^{bb}	AP^{bb}_{50}	AP^{bb}_{75}	AP^{bb}_{S}	AP^{bb}_{M}	AP^{bb}_{L}
R50-C4-1x	63.645	93.981	76.370	-	50.679	64.709
R50-DC5-1x	61.058	93.483	73.344	-	34.215	62.998
R50-FPN-1x	**66.551**	93.925	78.967	-	53.249	67.489
R50-C4-3x	64.719	97.138	76.584	-	40.502	66.834
R50-DC5-3x	59.767	92.146	72.093	-	46.909	61.354
R50-FPN-3x	64.067	92.018	78.886	-	54.923	64.864
R101-C4-3x	65.283	94.259	80.584	-	51.239	66.416
R101-DC5-3x	61.044	96.915	66.346	-	44.013	63.445
R101-FPN-3x	64.497	95.336	79.196	-	49.102	65.946
X101-FPN-32x8d-3x	66.305	94.651	79.458	-	54.543	67.229

Table 6.6: **Object Detection** *bounding box* AP on `ailaron-copepod-petridish-test` after training on `ailaron-copepod-petridish-train` for the backbones with available checkpoints for Mask R-CNN in Detectron2 framework.

Tables 6.6 and 6.7 show the object detection and instance segmentation mask accuracy results of the different models trained over *ailaron-copepod-petridish-test*.

Table 6.6 show the object detection accuracy from the experiments on *ailaron-copepod-petridish-test* after training on *ailaron-copepod-petridish-train*. We see a total and relative increase of 6.784 points and 11.4% respectively on the AP^{bb} from the worst to the best performing model. These are very small differences which means that all the models are performing almost equally well in terms of bounding box accuracy for object detection.

In terms of instance segmentation masks (table 6.7) however, there are major differences in performance between the models. We can see an extraordinary large total and relative improvements of 27.688 points and 206.1% respectively on the AP^{m} from the worst to the best performing models. We see that feature of the network has the largest impact on the AP^{m}. The C4 shows the weakest performance, the DC5 version shows significant improvement and the FPN shows even further improvement beyond the performance of the DC5, see chapter 3 for differences in the architectures.

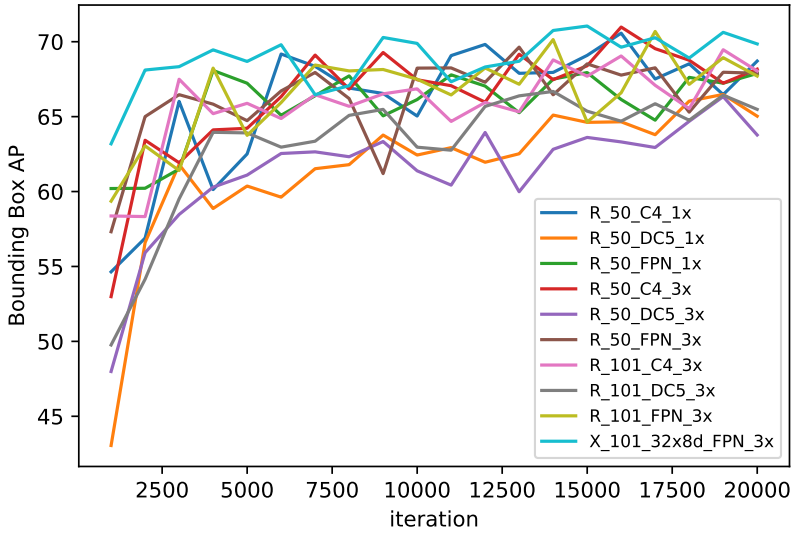


Figure 6.2: **Object Detection** *bounding box* AP on `ailaron-copepod-petridish-val` every evaluation step during training.

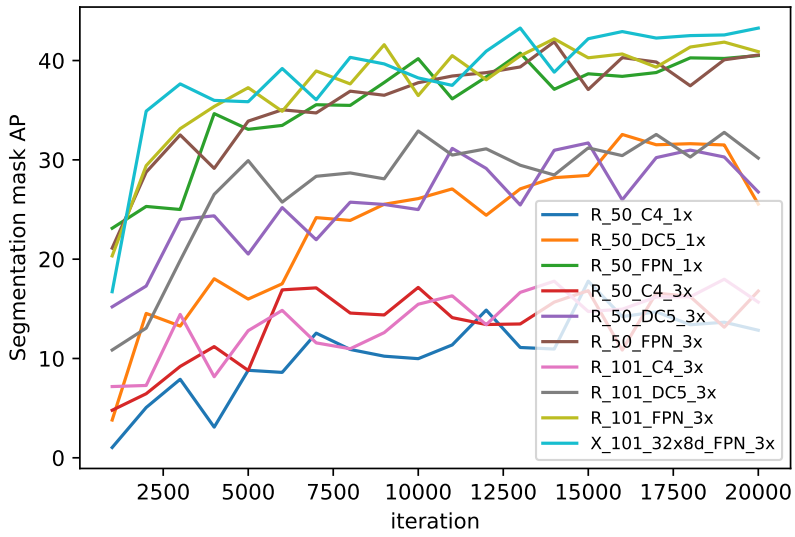


Figure 6.3: **Instance segmentation** *mask* AP on `ailaron-copepod-petridish-val` every evaluation step during training.

Backbone	AP^m	AP^m_{50}	AP^m_{75}	AP^m_S	AP^m_M	AP^m_L
R50-C4-1x	13.431	72.337	0.087	-	6.674	19.604
R50-DC5-1x	23.240	89.192	0.069	-	9.372	28.218
R50-FPN-1x	38.374	93.048	6.445	-	22.948	42.574
R50-C4-3x	17.153	87.881	0.083	-	6.750	23.762
R50-DC5-3x	25.504	91.408	0.459	-	13.628	31.386
R50-FPN-3x	38.491	94.254	3.542	-	24.217	43.762
R101-C4-3x	14.266	76.429	0.145	-	6.812	20.891
R101-DC5-3x	27.121	92.309	0.761	-	16.723	33.861
R101-FPN-3x	38.189	96.645	6.464	-	25.922	44.059
X101-FPN-32x8d-3x	**41.119**	97.843	7.628	-	28.604	46.139

Table 6.7: **Instance segmentation** *mask* AP on `ailaron-copepod-petridish-test` after training on `ailaron-copepod-petridish-train` for the backbones with available checkpoints for Mask R-CNN in Detectron2 framework.

The depth of the networks shows less of an impact on the performance though as we see the differences between the ResNet-101 and ResNet-50 backbones are very small. The findings are in line with that of He et al. (2017)[38], but the gaps in performance from the features are a lot larger in our results than that of the paper.

Table 6.8 shows the accuracy of models trained on the predictions output from the PySilCam[60]. The images used for training are the same as in the training part of the custom dataset. The trained models are evaluated against both the output from the PySilCam and the ground truths on the test set. This shows how well the model learns the ground truth, the output of the PySilCam, and how well this learning corresponds to actual the truth.

The annotations output from the PySilCam used for both training and testing are filtered based on object size. Any object with a bounding box area lower than 32^2 pixels are not included as annotations.

In fig. 6.4 and 6.5 we can see an example image from the test split of the ailaron-copepod-petridish dataset with some annotations overlaid. Both figures show the same

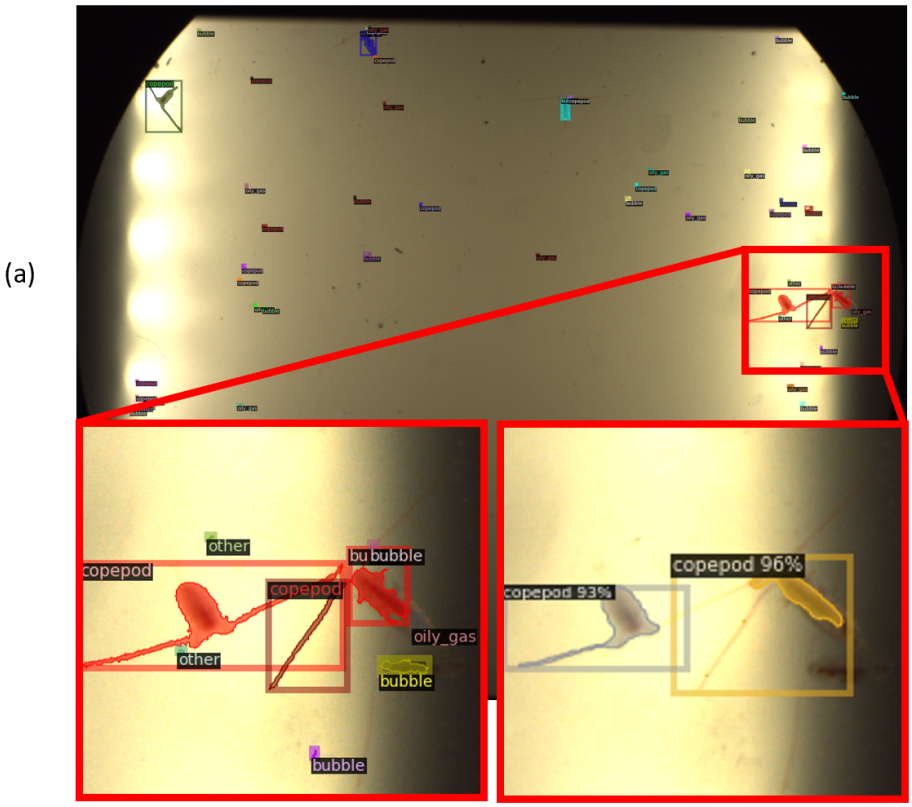


Figure 6.4: (a) Full image scene sample from `ailaron-copepod-petridish-test` with output from **PySilCam** visualized. (b) Zoom of PySilCam ouput visualized. (c) The Box and mask prediction outputs from the X101-FPN model presented in table 6.8.

Test set	AP^{bb}	AP^{bb}_{50}	AP^{bb}_{75}	AP^{m}	AP^{m}_{50}	AP^{m}_{75}
PySilCam output	11.033	21.328	9.229	9.040	19.141	8.307
`ailaron-copepod-petridish-train`	1.064	3.702	0.697	0.872	4.604	0.000

Table 6.8: **Instance segmentation** *mask* AP and **Instance segmentation** *bbox* AP from X101-FPN-32x8d-3x models trained over the segmentation result from the PySilCam[60] evaluated on the `ailaron-copepod-petridish-test`.

image but different annotations. Figure 6.4 shows poorly fitting annotations both for the bounding boxex and the masks. (a) shows a full image scene with the full output from the PySilCam framework after processing the image. It is clear that there are a lot of small boxes and masks that shouldn't be there, compared to fig. 6.5 (a) which has the manual annotations visualized (fig. 6.5 (b) is a zoom with the manual labels). We see that the predictions in fig. 6.5 (c) from the X101-FPN-32x8d-3x[3]model follow the ground truth very well. It is not perfect though, as the AP^m is not at 100. The zoom fig. 6.5 (c) which is prediction outputs from a different X101-FPN-32x8d-3x[4]does not follow what it has trained on, eg. (b) as well and we see from table 6.8 that it generally has very poor performance.

Summary

In table 6.9 we show the overall accuracy on object detection (table 6.6) and instance segmentation mask (table 6.7) together with the training and inference speed for the models.

In **training speed** we see a significant difference from the fastest to the slowest training time with a relative increase of 224.7% in terms of seconds per iteration. The relative increase in accuracy for object detection and segmentation mask from the fastest to the slowest training models are 2.5% and 139.7% respectively.

For **inference time** we see a relative increase of 123% from the fastest to the

[3]The model from table 6.6, 6.7 & 6.9
[4]The model from table 6.8

Backbone	training (s/iter)	inference (s/im)	AP^{bb}	AP^{m}
R50-C4-1x	0.1163	0.162453	63.645	13.431
R50-DC5-1x	0.2279	0.209720	61.058	23.240
R50-FPN-1x	0.1726	0.104279	**66.551**	38.374
R50-C4-3x	**0.1152**	0.224492	64.719	17.153
R50-DC5-3x	0.2293	0.140033	59.767	25.504
R50-FPN-3x	0.1777	0.110596	64.067	38.491
R101-C4-3x	0.1847	0.238032	65.283	14.266
R101-DC5-3x	0.2992	**0.100673**	61.044	27.121
R101-FPN-3x	0.2440	0.123223	64.497	38.189
X101-FPN-32x8d-3x	0.4893	0.163017	66.305	**41.119**

Table 6.9: Key performance metrics to compare across the different models in order to choose the model best suited for application. Same models as table 6.6 & table 6.7. All models are evaluated on `ailaron-copepod-petridish-test` after training on `ailaron-copepod-petridish-train`. For comparison to other datasets see the detectron Model Zoo[3].

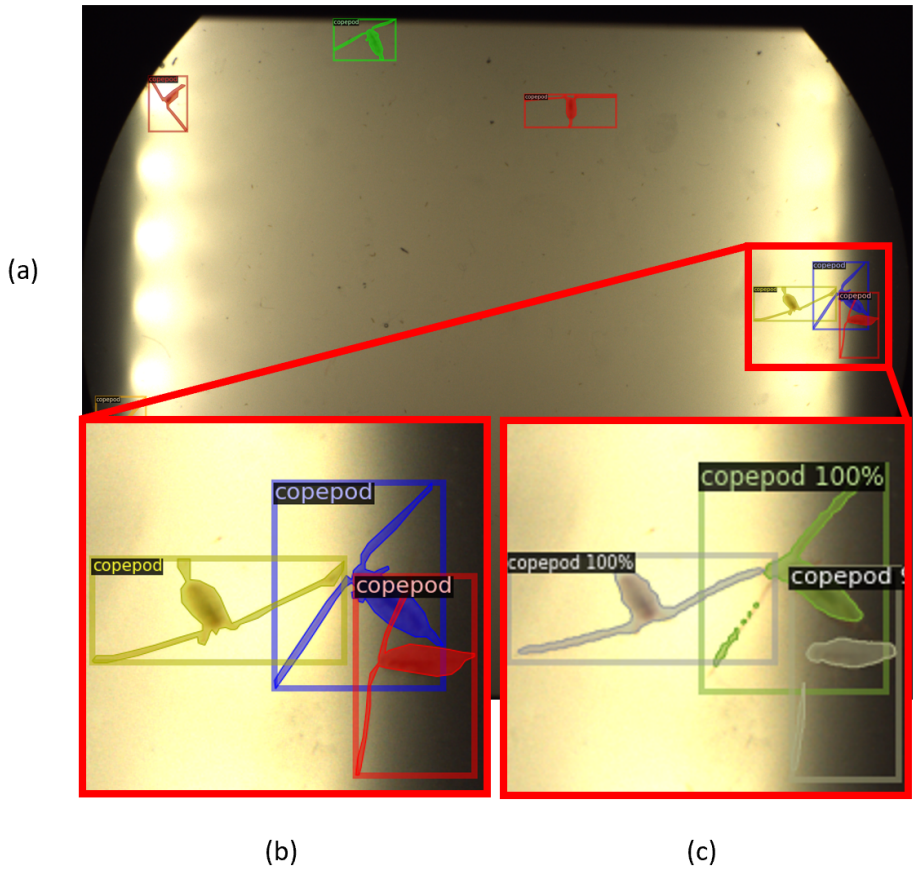


Figure 6.5: (a) Full image scene sample from `ailaron-copepod-petridish-test` with ground truth labels visualized. (b) Zoom of ground truth labels visualized. (c) The Box and mask prediction outputs from the X101-FPN model presented in table 6.9.

slowest model. The most accurate one in terms of segmentation, ResNeXt-101 with FPN, shows to be exactly halfway between those two models.

Chapter 7

Discussion

In this chapter, we discuss the different aspects associated with the results presented in chapter 6.

7.1 Confirming results on existing datasets

We only download and run inference on the trained models shown to be the best-performing ones by the authors, He et al.. It is important to note that we run the evaluation over the validation set `coco-minival-17`, not the `test-dev17` set as presented in the paper[38] presenting the method. This will cause some deviations from the results presented in the paper, which we see as a slight increase in all the metrics. This is not surprising as the training process isn't completely independent of the validation set which will cause the model to have a naturally better fit to this data than completely unseen data like the test set. It is in general considered bad practice to benchmark performance on data the model has interacted with during the training process, but we were left with no other choice as the framework wouldn't properly load the annotations for the test set. The results still show the essence of the findings, if not the same metric values.

7.2 Experiments on the custom planktonic dataset

It is important to note that the results presented here are on the ailaron-copepod-petridish data, a binary detection dataset which is a significant difference from the existing datasets like Coco[48] featuring upwards of one hundred classes.

Accuracy considerations for Mask R-CNN are twofold as it is doing both object detection like it's predecessor Faster R-CNN and instance segmentation in the second branch in the head of the architecture. In general, the variance in object detection accuracy among the models is so small compared to that of the segmentation that this is of little interest do discuss. The mask accuracy is varying a lot and the most accurate segmentation model is a clear winner over the other candidates.

Pre-training on the Coco dataset is performed on all the models we train ourselves as we start on a model checkpoint. It interesting to note that the different pre-training schedules don't seem to have any significant effect on the different backbones, especially the FPNs. The data used for pre-trained is quite different from the planktonic scenes. We start with the weights suited for the Coco data and fine-tuning them to the ailaron-copepod-petridish data. The 1x models have been trained on $\frac{1}{3}$ the number of epochs from the 3x, and the gain from the last $\frac{2}{3}$'s of the pre-training are not making a significant impact.

Training speed is of shrinking significance as computing power and available hardware is becoming less and less of an issue. This means that the training time is considered less important because it can be done on powerful GPU accelerated hardware designed for computationally heavy operations and the training of deep structures. In time-sensitive situations, it can still be an important metric to consider. If trade-offs have to be made training speed is likely to be the first one to be sacrificed if the improvement in other metrics is significant. Especially the increase in segmentation accuracy is more desired than low training time in this case. This is because training does not have to be done in-situ where hardware availability might be limited.

Certain hyperparameters like how many region proposals are generated and the batch size used will impact the time for one training iteration, while others like learning rate will affect how many iterations are required to reach a desirable performance.

In conjunction with tuning these parameters the total time to train the model can be changed. The experiments are all performed with the default value of region proposals connected to the backbone architectures in the implementation documentation[4], the same batch size, and the same base learning rate. Varying these values might produce different results from the ones presented here.

Inference time is of varying significance depending on the application of the system. This will the importance of a low inference time will depend on whether the data has to be processed in real-time and if so, how fast the samples are collected. The hardware available in the in-situ system is comparable to the hardware used for these experiments, so an inference stated in the results here will likely be very similar to what would be the reality in the in-situ system. There is some correlation between the features of the backbone architecture in terms of the total inference time per image, but less so connected to the depth of the architecture. Luckily there doesn't seem to be a positive correlation between inference time and accuracy, meaning we don't necessarily have to make a sacrifice with increased inference time to achieve better accuracy. Because we want to process images in real-time, we preferably need the inverse of the inference time to be lower than the sampling frequency to be able to keep up with the processing. The system is designed to run at approximately 5 samples per second, so an inference time of less than 0.2 seconds per image is to be desired. Not all the models fulfill the criteria, but most of them do, the one with the highest AP^m included.

The Weakly supervised experiment is using segmentation predictions from the pipeline of traditional methods. This is the closest comparison we do between the output from the PySilCam and the ground truths labeled manually. Implementing a process to evaluate this would have been very time consuming and was not set as a priority for this project. Comparing the results from a model trained on the manual "perfect" annotations (fig. 6.5 predictions (c) comapred to ground truth (b)) of the ailaron-copepod-petridish data and the weak predictions of the PySilCam will at least close in on quantifying the viability of the output from the old framework(fig. 6.4 predictions (c) compared to training truth (b)). The results are not directly comparable for the same reason the custom models are not comparable to that of other datasets.

We move from binary detection and segmentation to multi-class consideration. If we filter away all objects with a bounding box area of $< 32^2$, as we know there are no objects below that threshold, the predictions from the PySilCam detects a total of 627 objects. Only 316 of them are classified as Copepods by the pipeline. This results in a copepod precision of 50,4%. By inspection of the data however, we can determine that all objects that should be detected in this data are supposed to be copepods. This should hopefully illustrate the poor quality of these predictions by the PySilCam. Note that no threshold filtering on class confidence is considered. Object detection is determined by pixel clustering, and the object is classified into the category with the highest class score. From the results in table 6.8 we can see that these weren't very successful experiments. This made it clear that further efforts should either be unsupervised or we need better data to continue the supervised learning as using the PySilCam's output as annotations for a weakly supervised approach does not seem viable. Early attempts to train models using this approach and manually inspecting the results was what motivated the manual annotation and construction of the novel dataset.

Dataset format inconsistencies have been discovered in the manual construction of Coco format annotations and the automatic conversion from the Detectron default dictionary format. Unfortunately, the cause for this has not been uncovered. We consistently state the results from evaluating the coco format other than in table 6.5 because we also use the Detectron built-in Coco evaluator. It is very peculiar that this inconsistency arises as the Detectron default dictionaries are created from a coco format using custom-written code detailed in appendix B, then the framework converts it back again.

Summary

To summarize the impact of all of the metrics, if a "best model" is to be chosen for this application based on the results we have on the ailaron-copepod-petridish-test, it will be the ResNeXt-101-FPN model. This model did not shot the highest AP in the object detection, the the difference to the next model is only from 66305 to 66.551 points. This

is a total and relative increase of 0.246 and 0.37% respectively. The model is showing an impressive improvement of 2.628 points in AP^m (relative improvement of 6.8%) on the second most accurately segmenting model however, which is a significant increase. It is exactly in the center of the range of inference time but is by far the slowest model to train. The inference time is 0.163 s/im, which is fast enough as the framework is designed to run at approximately 5 frames per second and this model can handle 6 per second. The training time is a sacrifice worth making as the training is not something that has to be done frequently and the benefits in accuracy outweigh the downsides to increased training time by a long shot.

Chapter 8

Conclusions and future work

This chapter will discuss the conclusions that can be drawn from the findings and results presented previously in the thesis

Conclusion

The dataset constructed, the models trained and the results presented from the work in this thesis only provide a proof of concept. Showing that there seems to be significant promise in Mask R-CNN for this application. It is quite clear that the Mask R-CNN algorithm is well suited for the task of detecting and segmenting planktonic organisms in an image scene in real-time in an in-situ system compared to the one considered here. There is still potential for improvement if certain limitations to the project and future work are addressed.

Future Work

This section presents proposals for the next steps for the continuation of the work or possible improvements on the work presented in this thesis.

- Finalizing a module to embed in the PySilCam software suite as an alternative to the currently used modules is a natural task to consider. Most of the code necessary for the implementation exists in the files submitted as an attachment to this thesis and uploaded to the GitHub connected to it. This can be reorganized to provide such a module.

- Mask R-CNN is a general algorithm that is constructed by modular network architectures, concepts, and mechanisms. In this thesis, we have been using the implementation of the software framework Detectron2 by Facebook's Artificial Intelligence Research. This is a heavy dependency if only parts of the implementations are utilized in practice. After further results in what architectural configuration gives the best performance in the application at hand, looking at the possibility to develop a more lightweight custom implementation of the algorithm in another framework should be considered as this can lighten the memory overhead and dependencies required on the in-situ system.

- The Detectron2 framework contains a lot of tools and implementations of conceptual architectures intended to make it easy to implement new custom algorithms for research. Exploring the potential of these tools and implementing other promising algorithms to compare to the already implemented ones is a very relevant direction to take.

- The focus of the experiments in this thesis has been to give a broad analysis of the potential of mainly Mask R-CNN so little work has been put into the tuning of hyper-parameters used during training of the models. A more thorough analysis of the available configurations offered by the implementation and tuning of the hyper-parameters where the potential for improvement can be found is likely to improve upon the results shown in this thesis.

- The dataset constructed as a part of the contributions in this thesis only allowed binary detection and segmentation due to the inclusion of only one object class in the data. The number of data samples was also very limited due to the resources available for labeling. Increasing both the quality and quantity of data will be

necessary to perform a more detailed analysis of the performance of algorithms on this type of data.

- The software system currently performing object extraction, the PySilCam, is performing the segmentation after pre-processing the images in the form of background correction. All the experiments in this thesis have been conducted on raw unprocessed image scenes. Adapting the manual annotations to the background-corrected images and do training and evaluation over sets of those samples and compare the results to that of the models trained and evaluated on raw images could provide valuable insight in whether the background correction step is necessary or even beneficial.

Due to the work extended beyond this thesis for the conference submission in appendix A and the work promised there some of the points in the future work will be addressed after the thesis period. This mainly concerns the implementation of a module for the PySilCam software suite and the evaluation of Mask R-CNN's performance on the background-corrected images as these are topics related to the submission.

Increasing the amount of annotated data is also being addressed as a marine biologist has been hired for the task of annotating some of the stored data.

Appendix A

OCEANS abstract

Automatic in-situ instance and semantic segmentation of planktonic organisms using Mask R-CNN

Sondre Bergum
dept. of Engineering Cybernetics
NTNU
Trondheim, Norway
sondreab@stud.ntnu.no

Aya Saad
dept. of Engineering Cybernetics
NTNU
Trondheim, Norway
aya.saad@ntnu.no

Annette Stahl
dept. of Engineering Cybernetics
NTNU
Trondheim, Norway
annette.stahl@ntnu.no

***Index Terms*—in situ particle analysis, deep learning, semantic segmentation, plankton taxa distribution, mask r-cnn**

***Abstract*—Planktonic species are one of the most numerous organisms on the planet and form the basis for the ecological food chain in the ocean, making them a key component of aquatic ecosystems. These organisms are susceptible to environmental changes, and studying their temporal variation in spatial abundance and taxa distribution plays an integral part in understanding and predicting the development of ecosystems in the ocean. The study of these planktonic organisms has been limited by manual analysis until recent years where several efforts of developing automatic imaging systems to aid scientists in gathering the necessary data [1, 2, 3].**

Davies et al. (2017) [4] (PySilCam Suite[5]) developed a particle imaging system, that features traditional methods of computer vision for pre-processing and segmentation of the input images. Particles are extracted based on a binary threshold mask and classified by a deep convolutional neural network (DNN) to build a distribution of the particles extracted from the images. The location of the extracted particle is saved as the objects' bounding box. The evaluation of the performance of this framework motivates finding alternative methods of analyzing the images to improve upon the measurement of planktonic distribution and concentration.

Our contribution is a novel framework using deep learning instanced semantic segmentation for in-situ imaging and particle extraction, calculating their distribution and concentration. As part of this contribution, we have developed a manually annotated dataset for segmentation to train the deep learning architecture in our framework.

The proposed system substitutes the particle segmentation, extraction, and classification with a fully supervised deep learning method capable of object detection, localization, semantic, and instanced segmentation. The architecture used is detectron2[6], an extension of the Mask R-CNN[7] which is the next iteration of Fast and Faster R-CNN[8, 9]. This method has been chosen as its backbone architecture and method has provided the foundation for multiple winning contenders in both the ILSVRC [10] and COCOC [11] competitions as well as achieving state-of-the-art results on the PASCAL VOC[12] among other widely used datasets.

The framework (fig. 1) starts by performing pre-processing on each image with a moving average background correction using images recorded prior to the raw image being processed[4]. The background-corrected image is then used as the input for the segmentation process. The inferred data from the instance segmentation is then used to build the particle distribution and output images with the bounding boxes and segmentation masks of each object visualized.

As a requirement for the framework to function appropriately, training data of sufficient quality is necessary. As part of the work, we compile a novel dataset with complete bounding boxes and instanced semantic segmentation masks from images captured in a lab environment. The dataset is a key contribution, as even though multiple publicly available datasets containing microscopic images of particles and planktonic life forms exist, like, for instance[13], they are classification sets and not suited for supervised segmentation. Our novel dataset is to the best of our knowledge, the first publicly available, manually annotated planktonic dataset suited for the task of supervised instanced semantic segmentation.

We benchmark the performance of previous efforts of traditional segmentation compared to our novel framework using the annotated dataset as ground truth to gauge the difference in performance. The perfomance of a deep learning method will depend on its model architecture and learnable parameters. We train multiple models using both our annotated dataset (a) and post-processed classification and segmentation output from the PySilCam Suite (b) as training data and evaluate and compare the different models performance. We also quantify the impact of using pre-processed background-corrected images (d) or the raw images (c) as input to the framework. Indecies (a)-(d) are referenced in fig. 1. Evaluation results of the deep learning architecture adopted in our framework show better performance in terms of speed and accuracy on captured images in-situ compared to the system developed in the PySilCam Suite by Davies et al. (2017) [4] based on traditional segmentation methods. Moreover, the network achieved better identification and classification of objects when the corrected image (d) replaced the raw image (c) input to the workflow. From fig. 2-3 a copepod was identified with a higher probability of 98% and as one entity by our framework, compared to 88.9% from the PySilCam Suite. Furthermore, a significant improvement in the object classification was clear when we train the model for our framework over our novel labeled dataset before the in-situ operations.

The complete framework is embedded in an autonomous lightweight underwater vehicle (LAUV). Its purpose is to provide data for the construction of a dynamic probability density map from the real-time identification and classification of plankton taxa.

This research is funded by RCN FRINATEK IKTPLUSS program (project number 262701) and supported by NTNU AMOS.

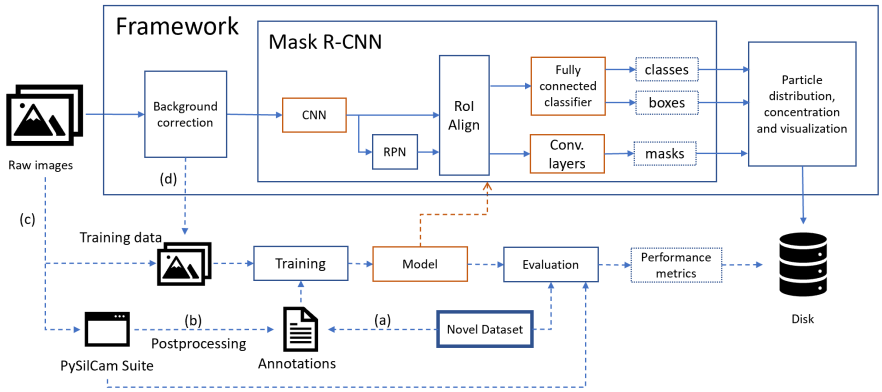


Fig. 1. A simple topological overview of our framework. The background correction module is from the PySilCam suite, the Mask R-CNN is the deep learning method embedded in the framework. The dotted lines represent processes performed as a necessary preparation steps before the framework is put to use in-situ. The pretrained model is embeded into the framework and the learnable parameters impacted by this process are indicated in orange.

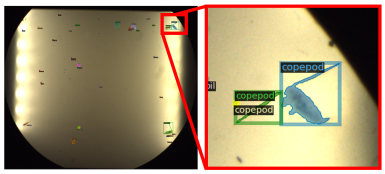


Fig. 2. From left to right: the raw captured image from the camera and a zoomed cropped identified copepod. Overlay of unfiltered data of class, bounding box and binary threshold mask output from the PySilCam Suite are shown on the raw image. The class is determined by the highest classification probability. Clearly, the antennas are identified as unique copepod instances.

Fig. 3. From left to right: corrected background image and a zoomed cropped copepod of the framework end result. Overlay of inferred class, bounding box and segmentation mask from detectron2. Clearly, the copepod object is identified as one instance.

References

[1] H. M. Sosik and R. J. Olson, "Automated taxonomic classification of phytoplankton sampled with imaging-in-flow cytometry," *Limnology and Oceanography: Methods*, vol. 5, no. 6, pp. 204–216, 2007.

[2] H. Bi, Z. Guo, M. C. Benfield, C. Fan, M. Ford, S. Shahrestani, and J. M. Sieracki, "A semi-automated image analysis procedure for in situ plankton imaging systems," *PloS one*, vol. 10, no. 5, 2015.

[3] A. Saad, E. Davies, and A. Stahl, "Recent advances in visual sensing and machine learning techniques for in-situ plankton-taxa classification," presented at Ocean Sciences Meeting 2020, San Diego, CA, 16-21 Feb., 2020, 636384.

[4] E. J. Davies, P. J. Brandvik, F. Leirvik, and R. Nepstad, "The use of wide-band transmittance imaging to size and classify suspended particulate matter in seawater," *Marine pollution bulletin*, vol. 115, no. 1-2, pp. 105–114, 2017.

[5] "Pysilcam suite," https://github.com/SINTEF/PySilCam, [Online; Accessed 30-April-2020].

[6] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick, "Detectron2," https://github.com/facebookresearch/detectron2, 2019.

[7] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[8] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

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

[10] "Imagenet large scale visual recognition challenge," http://image-net.org/challenges/LSVRC/.

[11] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.

[12] M. Everingham, S. A. Eslami, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes challenge: A retrospective," *International journal of computer vision*, vol. 111, no. 1, pp. 98–136, 2015.

[13] H. M. Sosik, E. E. Peacock, and E. F. Brownlee, "labeled ifcb images," https://hdl.handle.net/1912/7350, 2014.

Appendix B

Code

In this chapter we will provide some documentation for the code submitted in conjunction with the thesis which has been made available on `https://github.com/AILARON/Segmentation`[14]. This will not be a full documentation and the code quality is very bad, so use at your own risk! All the code has been used with Jupyter Notebooks and is structured in such a way that using the cell structure in Jupyter is necessary if the code is to be run without major refactoring. If a function isn't covered for a file it means it has nearly the same implementation as in a previously covered file.

Unless otherwise stated this is code written by the author of this thesis.

B.1 build_dataset.py

This file contains functions and code for importing the information in *.csv and export files output from the PySilCam and building Detectron2 standard dictionary dataset. It has dependencies on `https://github.com/facebookresearch/detectron2`[77] and `https://github.com/emlynjdavies/PySilCam`[60].

- `def extract_pixels(im, bbox)`

This function binary image (im) as a nd-array of a binary segmentation mask and a bounding box [x_min, y_min, x_max, y_max] on, list of 4 integers XYXY_abs format see section 4.2 and returns two arrays containing the x and y coordinates of any countour detected within the bounding box.

- `def read_stats(directory = DIRECTORY)`

Reads a *.csv stat file from PySilCam output with path directory and returns it as a dictionary.

- `def build_annotation_dictionary(directory = DIRECTORY, size_threshold = 0)`

Takes in the root directory of a PySilCam data folder with subfolders 'proc', 'raw', export, etc., reads the *-STATS.csv file and returns a Detectron2 standard dataset dictionary from the csv file and images in the subfolders. It will only include objects with a size_threshold < 'equivalent_diameter'. Parts of this function is based parts on the tutorial found at `https://colab.research.google.com/drive/16jcaJoc6bCFAQ96jDe2HwtXj7BMD_-m5`

- `def create_json_file(data, file_name, directory=EXPORT_DIR):`

Json-serializes and saves data as directory/file_name.json.

- `def read_json_file(file_name, directory=EXPORT_DIR)`

Loads directory/file_name.json and returns the contents of the file. This function assumes it is reading a Detectron2 standard dictionary formated file as it tries to add a non-JSON-serializable field to every segmentation object in the dictionary before returning it.

- `def save_dataset_visualization(dataset, directory=VISUALIZE_DIR)`

  Loads the dataset with name 'dataset' which is assumed to already be registered in the Detectron2 dictionary database and the dataset dictionary file can be found in 'directory'. It will load all the segmentation annotations from the dataset and visualize them on the images and save the images in the dataset with visualizations to 'directory'.

B.2 dataset_training.py

This file contains code and functions used for training Detectron2 models. Dependent on `https://github.com/facebookresearch/detectron2`[77]

- `class LossEvalHook(HookBase)`
  `class MyTrainer(DefaultTrained):`

  These are a custom classes from `https://gist.github.com/ortegatron/c0dad15e49c2b74de8bb09a5615d9f6b`, instantiating some of the default classes in Detectron2 adding evaluation hooks in the trained in order to report validation loss.

- `def train_dataset(dataset):`

  This function is deprecated, but the actual implementation doing what it is intended to do is found in a cell further down. Define 'task' and 'arch_backbone' to fit a config file at the Detectron2 github [77] and train after defining parameters. Code for evaluating a dataset is further down in the same cell.

- `def inference(dataset,`
  `output_path = OUTPUT_PATH,`
  `inference_dir = os.path.join(DIRECTORY,`

```
                                                INFERENCE_DIR),
                weights = "model_final.pth")
```

This function passes every file in a dataset through a model and infers predictions, and saves the prediction visualizations on the images to 'inference_dir'. There are some difficulties loading models into the current config cfg, but if a model was trained and is still in memory it will not be a problem.

- ```
  def inference_over_directory(dataset,
                               files_dir,
                               output_path = OUTPUT_PATH,
                           inference_dir = os.path.join(DIRECTORY,
                                                        INFERENCE_DIR)):
  ```

  This function will pass all files in 'files_dir' through the currently loaded cfg model and save the images with visualized predictions to 'inference_dir'. It is supposed to load a model checkpoint from 'output_path' but the model loading isn't working as intended.

In this file there is also code for registering datasets both on the detectron format and the Coco format.

B.3 Utilities.py

Some of the code in here should maybe be in the more aptly named build_dataset.py as a lot of the functions in here does that, but it is not.

- ```
  def clean_vgg_annotator_coco_file(file_name, directory)
  ```

  Takes 'file_name'+'.json' file assumed to be a coco dataset dictionary file and adapts it to the style with fields for the ailaron-copepod-petridish-dataset[14], saves a new file as 'file_name'+'clean.json' and returns the dataset dictionary.

- `def extract_sample_list_from_coco_set(file_name, directory)`

  Returns the a list of files contained in the dataset file 'file_name'+'.json' and saves it as 'file_name'+'_files.json'.

- `def split_sample_list(file_name, directory)`

  Splits the list of filenames in 'file_name'+'.json' roughly into 15%-70%-15% splits at random, returns the dictionary and saves it as 'file_name'+'_split_sets.json'.

- `split_coco_set(dataset_file_name, split_file_name, directory)`

  Splits the dataset in 'dataset_file_name'+'.json' into the train, validation and test set lists specified by 'split_file_name'+'.json', returns and saves the dataset dictionaries as 'dataset_file_name'+'_'+{'train','val','test'}+'.json'

- `def from_coco_to_detectron_default_dataset(coco_dataset, coco_directory, image_directory)`

  Converts a coco format dataset into a Detectron2 default dictionary dataset format. This is redundant as I found out that there is a build-in function for this in the Detectron2 framework.

- `def from_detectron_to_ \...\ coco_default_dataset(detectron_dataset, dataset_directory, area_threshold = 0)`

COnverts a detectron2 dataset dictinoary into a coco dataset filtering away all objects with a boundingbox area < 'area_threshold'

The remaining cells in this file are filled with function calls in order to manipulate dataset dictionary files.

B.4 metric_plotting.py

File for plotting the results coming out of the 'metric.json' files output from Detectron2 during training.

- `def load_json_arr(json_path)`

  Loads the metrics as a list of dictionaries.

- `def load_files_in_dir(directory = "./")`

  Returns two lists containing the names of all the files in 'directory'

- ```
  def plot_all_models(models = [],
                      files = [],
                    saveto = "/home/sondreab/Desktop/METRICS/test",
                      figname = 'fig.pdf',
                      metric = "validation loss"):
  ```

  Plos the metric put in as a parameter from all 'files' for all 'models' and saves into a plot at 'saveto/figname'.
  ```

Appendix C

Extended results

This chapter contains extended results considered too expansive or not relevant enough to include in the main chapters. For the main results see chapter 6.

Evaluation of Mask R-CNN on MS COCO

These are he extended results from running evaluation of trained models in Mask R-CNN[38] over the `coco-minival-2017` dataset. For details on the model architectures see section 3.3 and for explanation of the metrics see section 5.5.

ResNet101-C4-3x

Evaluation results for bbox:

```
|   AP   |  AP50  |  AP75  |  APs   |  APm   |  APl   |
|:------:|:------:|:------:|:------:|:------:|:------:|
| 42.576 | 62.121 | 46.048 | 23.162 | 47.069 | 57.639 |

| category      | AP    | category     | AP    | category       | AP    |
|:--------------|:------|:-------------|:------|:---------------|:------|
```

person	55.793	bicycle	33.369	car	43.746
motorcycle	45.541	airplane	67.715	bus	67.597
train	65.477	truck	35.952	boat	28.425
traffic light	25.765	fire hydrant	68.869	stop sign	66.963
parking meter	46.942	bench	25.275	bird	37.139
cat	69.284	dog	59.732	horse	61.445
sheep	50.008	cow	55.384	elephant	64.648
bear	68.836	zebra	68.947	giraffe	69.047
backpack	17.493	umbrella	41.280	handbag	15.435
tie	34.084	suitcase	38.853	frisbee	65.146
skis	26.679	snowboard	39.239	sports ball	42.227
kite	40.853	baseball bat	25.991	baseball glove	36.577
skateboard	54.755	surfboard	40.161	tennis racket	51.048
bottle	38.686	wine glass	36.331	cup	42.050
fork	39.993	knife	18.808	spoon	21.705
bowl	42.857	banana	24.906	apple	21.718
sandwich	37.041	orange	30.550	broccoli	23.024
carrot	22.378	hot dog	34.065	pizza	53.451
donut	47.440	cake	35.944	chair	29.403
couch	44.349	potted plant	28.041	bed	40.052
dining table	29.840	toilet	63.560	tv	58.792
laptop	63.260	mouse	58.099	remote	29.897
keyboard	54.069	cell phone	35.746	microwave	61.308
oven	33.899	toaster	46.902	sink	36.683
refrigerator	60.299	book	13.114	clock	49.489
vase	37.745	scissors	30.014	teddy bear	49.586
hair drier	5.759	toothbrush	23.501		

Evaluation results for segm:

AP	AP50	AP75	APs	APm	APl

| 36.652 | 58.478 | 39.256 | 15.839 | 40.290 | 54.823 |

category	AP	category	AP	category	AP
person	45.734	bicycle	19.551	car	39.378
motorcycle	34.742	airplane	45.709	bus	63.672
train	63.087	truck	34.226	boat	23.362
traffic light	25.004	fire hydrant	63.772	stop sign	64.512
parking meter	46.996	bench	17.147	bird	27.674
cat	65.552	dog	54.904	horse	39.663
sheep	41.469	cow	43.474	elephant	55.869
bear	66.466	zebra	56.116	giraffe	48.432
backpack	16.252	umbrella	44.665	handbag	13.300
tie	28.793	suitcase	40.306	frisbee	61.185
skis	1.942	snowboard	22.081	sports ball	41.729
kite	26.691	baseball bat	20.424	baseball glove	36.327
skateboard	30.744	surfboard	30.891	tennis racket	53.617
bottle	36.332	wine glass	30.271	cup	41.460
fork	17.283	knife	12.066	spoon	12.699
bowl	38.500	banana	20.759	apple	20.016
sandwich	38.064	orange	29.229	broccoli	22.222
carrot	19.124	hot dog	28.264	pizza	50.951
donut	46.334	cake	36.354	chair	19.140
couch	35.267	potted plant	22.624	bed	30.967
dining table	16.120	toilet	56.923	tv	60.631
laptop	59.277	mouse	56.867	remote	24.954
keyboard	51.321	cell phone	31.578	microwave	60.731
oven	32.172	toaster	51.083	sink	34.500
refrigerator	59.054	book	7.314	clock	49.840
vase	35.931	scissors	18.337	teddy bear	43.655
hair drier	2.626	toothbrush	15.858		

ResNet101-FPN-3x

Evaluation results for bbox:

AP	AP50	AP75	APs	APm	APl
42.928	63.323	46.834	26.396	46.595	56.127

category	AP	category	AP	category	AP
person	56.564	bicycle	33.320	car	46.299
motorcycle	45.460	airplane	64.943	bus	67.283
train	63.870	truck	37.436	boat	30.259
traffic light	28.938	fire hydrant	69.259	stop sign	69.186
parking meter	47.573	bench	25.150	bird	38.127
cat	69.155	dog	62.371	horse	58.714
sheep	54.080	cow	56.453	elephant	62.169
bear	72.848	zebra	67.389	giraffe	67.861
backpack	19.134	umbrella	39.688	handbag	17.461
tie	38.011	suitcase	40.343	frisbee	65.476
skis	27.408	snowboard	37.717	sports ball	48.816
kite	43.435	baseball bat	28.747	baseball glove	38.689
skateboard	55.744	surfboard	38.965	tennis racket	50.825
bottle	41.597	wine glass	38.114	cup	44.020
fork	38.213	knife	21.140	spoon	20.224
bowl	42.897	banana	24.497	apple	21.714
sandwich	36.331	orange	31.943	broccoli	23.172
carrot	23.027	hot dog	36.191	pizza	52.115
donut	46.332	cake	37.851	chair	29.237
couch	42.566	potted plant	27.836	bed	42.395

dining table	29.027	toilet	59.803	tv	57.551
laptop	62.153	mouse	63.043	remote	34.054
keyboard	51.766	cell phone	37.615	microwave	55.712
oven	34.679	toaster	37.318	sink	38.235
refrigerator	56.564	book	16.775	clock	52.138
vase	39.835	scissors	27.648	teddy bear	46.279
hair drier	3.819	toothbrush	23.679		

Evaluation results for segm:

AP	AP50	AP75	APs	APm	APl
38.629	60.449	41.271	19.484	41.324	55.289

category	AP	category	AP	category	AP
person	48.656	bicycle	19.881	car	42.382
motorcycle	35.045	airplane	51.234	bus	65.135
train	62.638	truck	36.306	boat	25.069
traffic light	27.870	fire hydrant	64.924	stop sign	68.890
parking meter	48.242	bench	18.392	bird	32.358
cat	67.826	dog	59.535	horse	41.996
sheep	46.230	cow	48.225	elephant	56.882
bear	70.021	zebra	58.701	giraffe	51.385
backpack	18.037	umbrella	46.775	handbag	16.628
tie	35.347	suitcase	42.334	frisbee	64.217
skis	4.251	snowboard	21.587	sports ball	48.371
kite	31.612	baseball bat	25.290	baseball glove	41.278
skateboard	33.579	surfboard	31.947	tennis racket	56.314
bottle	39.788	wine glass	33.805	cup	43.778
fork	18.271	knife	13.514	spoon	13.922
bowl	39.950	banana	19.389	apple	20.794

sandwich	37.969	orange	31.768	broccoli	22.995
carrot	19.378	hot dog	30.518	pizza	50.659
donut	46.410	cake	37.695	chair	19.690
couch	36.093	potted plant	23.250	bed	33.749
dining table	17.044	toilet	59.537	tv	60.378
laptop	60.789	mouse	63.925	remote	32.339
keyboard	50.498	cell phone	36.643	microwave	56.980
oven	32.181	toaster	40.386	sink	36.194
refrigerator	58.806	book	10.848	clock	51.760
vase	38.709	scissors	22.377	teddy bear	44.713
hair drier	2.207	toothbrush	15.221		

RexNeXt101-32x8d-FPN-3x

Evaluation results for bbox:

AP	AP50	AP75	APs	APm	APl
44.275	64.463	48.618	27.530	47.637	56.698

category	AP	category	AP	category	AP
person	57.651	bicycle	35.388	car	47.308
motorcycle	48.320	airplane	68.875	bus	68.867
train	64.160	truck	38.008	boat	29.573
traffic light	29.627	fire hydrant	70.423	stop sign	70.041
parking meter	49.111	bench	28.519	bird	39.758
cat	68.940	dog	65.395	horse	61.222
sheep	53.970	cow	57.410	elephant	63.595
bear	71.708	zebra	66.233	giraffe	66.055
backpack	18.641	umbrella	42.009	handbag	17.706
tie	37.690	suitcase	43.229	frisbee	68.790

skis	28.245	snowboard	40.892	sports ball	50.146
kite	44.481	baseball bat	34.012	baseball glove	39.575
skateboard	55.739	surfboard	41.350	tennis racket	52.942
bottle	41.044	wine glass	40.146	cup	46.715
fork	42.487	knife	24.409	spoon	22.919
bowl	42.706	banana	26.665	apple	24.040
sandwich	37.897	orange	31.677	broccoli	21.495
carrot	23.065	hot dog	38.699	pizza	54.799
donut	47.713	cake	36.572	chair	31.568
couch	44.605	potted plant	29.309	bed	40.774
dining table	29.508	toilet	61.194	tv	57.813
laptop	63.795	mouse	63.634	remote	37.507
keyboard	53.320	cell phone	38.375	microwave	59.853
oven	35.148	toaster	41.602	sink	38.742
refrigerator	57.462	book	16.950	clock	50.473
vase	38.557	scissors	27.986	teddy bear	49.778
hair drier	5.410	toothbrush	29.978		

Evaluation results for segm:

AP	AP50	AP75	APs	APm	APl
39.520	61.696	42.570	20.680	41.967	56.520

category	AP	category	AP	category	AP
person	49.691	bicycle	20.808	car	43.570
motorcycle	37.281	airplane	52.520	bus	67.027
train	63.167	truck	37.239	boat	26.006
traffic light	28.534	fire hydrant	65.609	stop sign	68.616
parking meter	48.877	bench	20.558	bird	33.292
cat	66.861	dog	61.694	horse	44.843

sheep	47.109	cow	48.363	elephant	58.010
bear	69.402	zebra	57.214	giraffe	49.374
backpack	18.661	umbrella	48.087	handbag	16.847
tie	34.860	suitcase	45.394	frisbee	66.896
skis	4.525	snowboard	24.961	sports ball	49.529
kite	31.851	baseball bat	28.397	baseball glove	41.354
skateboard	34.180	surfboard	34.324	tennis racket	56.721
bottle	39.195	wine glass	36.178	cup	45.935
fork	21.550	knife	16.179	spoon	15.688
bowl	38.990	banana	21.918	apple	23.024
sandwich	38.252	orange	31.746	broccoli	20.692
carrot	19.348	hot dog	30.440	pizza	52.075
donut	46.983	cake	35.752	chair	21.930
couch	36.845	potted plant	24.326	bed	32.816
dining table	17.269	toilet	58.858	tv	60.011
laptop	62.627	mouse	62.989	remote	34.564
keyboard	51.869	cell phone	37.269	microwave	58.977
oven	32.504	toaster	44.622	sink	37.177
refrigerator	58.898	book	11.750	clock	51.221
vase	38.320	scissors	21.809	teddy bear	46.552
hair drier	4.088	toothbrush	18.137		

Appendix D

Dataset formats

COCO format

```
Dataset Dictionary:

    {
        "info"          : info,
        "images"        : [image],
        "annotations"   : [annotation],
        "licenses"      : [license],
        "categories"    : [category],
    }

Fields:

    info{
        "year"          : int,
        "version"       : str,
        "description"   : str,
```

```
    "contributor"   : str,
    "url"           : str,
    "date_created"  : datetime,
    }

image{
    "id"            : int,
    "width"         : int,
    "height"        : int,
    "file_name"     : str,
    "license"       : int,
    "flickr_url"    : str,
    "coco_url"      : str,
    "date_captured" : datetime,
}

annotation{
    "id"            : int,
    "image_id"      : int,
    "category_id"   : int,
    "segmentation"  : RLE or [polygon],
    "area"          : float,
    "bbox"          : [x,y,width,height],
    "iscrowd"       : 0 or 1,
}

licence{
    "id"            : int,
    "name"          : str,
    "url"           : str,
}
```

```
category{
    "id"            : int,
    "name"          : str,
    "supercategory" : str,
}
```

References

[1] Berkley, university of california. https://www.berkeley.edu/. [Online; Accessed 25-June-2020].

[2] Ms coco detection evaluation metrics. https://cocodataset.org/#detection-eval.

[3] Detectron2 model zoo. https://github.com/facebookresearch/detectron2/blob/master/MODEL_ZOO.md, . [Online; Accessed: 5-July-2020].

[4] Detectron2 config references. https://detectron2.readthedocs.io/modules/config.html#config-references, . [Online; Accessed: 29-June-2020].

[5] Detectron2 built-in evaluators. https://detectron2.readthedocs.io/modules/evaluation.html#detectron2.evaluation.DatasetEvaluator, . [Online; Accessed: 27-May-2020].

[6] Detectron2 standard dataset dictionaries. https://detectron2.readthedocs.io/tutorials/datasets.html, . [Online; Accessed: 24-May-2020].

[7] Precision and recall. https://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html. [Online; Accessed 3-February-2020].

[8] Torch github readme. https://github.com/torch/torch7/blob/master/README.md, 2018. [Online; Accessed: 29-March-2020].

[9] AILARON. The ailaron project. https://www.ntnu.edu/web/ailaron. [Online; Accessed 28-October-2019].

[10] Nasullah Khalid Alham, Maozhen Li, Yang Liu, and Suhel Hammoud. A mapreduce-based distributed svm algorithm for automatic image annotation. *Computers & Mathematics with Applications*, 62(7):2801–2811, 2011.

[11] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.

[12] Lamberto Ballan, Tiberio Uricchio, Lorenzo Seidenari, and Alberto Del Bimbo. A cross-media model for automatic image annotation. In *Proceedings of International Conference on Multimedia Retrieval*, pages 73–80, 2014.

[13] Sondre A. Bergum. Msc preproject - advanced techniques for image segmentation to identify plankton-taxa and their distribution scope, 2019.

[14] Sondre A. Bergum. 2020 labeled ailaron-copepod-petridish dataset. https://github.com/AILARON/Segmentation, 2020.

[15] Dimitri P Bertsekas. Nonlinear programming. *Journal of the Operational Research Society*, 48(3):334–334, 1997.

[16] Hongsheng Bi, Zhenhua Guo, Mark C Benfield, Chunlei Fan, Michael Ford, Suzan Shahrestani, and Jeffery M Sieracki. A semi-automated image analysis procedure for in situ plankton imaging systems. *PloS one*, 10(5), 2015.

[17] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[18] Erik Bochinski, Ghassen Bacha, Volker Eiselein, Tim JW Walles, Jens C Nejstgaard, and Thomas Sikora. Deep active learning for in situ plankton classification. In *International Conference on Pattern Recognition*, pages 5–15. Springer, 2018.

[19] Augustin Cauchy. Méthode générale pour la résolution des systemes d'équations simultanées. *Comp. Rend. Sci. Paris*, 25(1847):536–538, 1847.

[20] Ronan Collobert, Samy Bengio, and Johnny Mariéthoz. Torch: a modular machine learning software library. Technical report, Idiap, 2002.

[21] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[22] Jifeng Dai, Kaiming He, Yi Li, Shaoqing Ren, and Jian Sun. Instance-sensitive fully convolutional networks. In *European Conference on Computer Vision*, pages 534–549. Springer, 2016.

[23] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. In *Advances in neural information processing systems*, pages 379–387, 2016.

[24] Emlyn John Davies, Per Johan Brandvik, Frode Leirvik, and Raymond Nepstad. The use of wide-band transmittance imaging to size and classify suspended particulate matter in seawater. *Marine pollution bulletin*, 115(1-2):105–114, 2017.

[25] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *International conference on machine learning*, pages 647–655, 2014.

[26] Abhishek Dutta, Ankush Gupta, and A Zissermann. Vgg image annotator (via). *URL: http://www. robots. ox. ac. uk/˜ vgg/software/via*, 2016.

[27] Mark Everingham, SM Ali Eslami, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes challenge: A retrospective. *International journal of computer vision*, 111(1):98–136, 2015.

[28] Kunihiko Fukushima. Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural networks*, 1(2):119–130, 1988.

[29] Alberto Garcia-Garcia, Sergio Orts-Escolano, Sergiu Oprea, Victor Villena-Martinez, and Jose Garcia-Rodriguez. A review on deep learning techniques applied to semantic segmentation. *TPAMI*, 2017.

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

[31] Ross Girshick, Ilija Radosavovic, Georgia Gkioxari, Piotr Dollár, and Kaiming He. Detectron. `https://github.com/facebookresearch/detectron`, 2018.

[32] Saurabh Gupta, Ross Girshick, Pablo Arbeláez, and Jitendra Malik. Learning rich features from rgb-d images for object detection and segmentation. In *European conference on computer vision*, pages 345–360. Springer, 2014.

[33] Jun Han and Claudio Moraga. The influence of the sigmoid function parameters on the speed of backpropagation learning. In *International Workshop on Artificial Neural Networks*, pages 195–201. Springer, 1995.

[34] Allan Hanbury. A survey of methods for image annotation. *Journal of Visual Languages & Computing*, 19(5):617–627, 2008.

[35] Bharath Hariharan, Pablo Arbeláez, Ross Girshick, and Jitendra Malik. Simultaneous detection and segmentation. In *European Conference on Computer Vision*, pages 297–312. Springer, 2014.

[36] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1904–1916, 2015.

[37] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[38] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[39] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[40] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7310–7311, 2017.

[41] ImageNet large scale visual recognition challenge. Imagenet large scale visual recognition challenge. `http://image-net.org/challenges/LSVRC/`.

[42] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678, 2014.

[43] Haijie Tian Yong Li Yongjun Bao Zhiwei Fang Hanqing Lu Jun Fu, Jing Liu. Dual attention network for scene segmentation. 2019.

[44] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[45] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[46] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521 (7553):436–444, 2015.

[47] Moshe Leshno, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867, 1993.

[48] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[49] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

[50] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[51] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[52] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3, 2013.

[53] Francisco Massa and Ross Girshick. maskrcnn-benchmark: Fast, modular reference implementation of Instance Segmentation and Object Detection algorithms in PyTorch. `https://github.com/facebookresearch/maskrcnn-benchmark`, 2018. Accessed: 25-March-2020.

[54] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[55] Rei Morikawa. 24 best image annotation tools for computer vision. `https://lionbridge.ai/articles/image-annotation-tools-for-computer-vision/`, 2019.

[56] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[57] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[58] Pedro O Pinheiro, Tsung-Yi Lin, Ronan Collobert, and Piotr Dollár. Learning to refine object segments. In *European Conference on Computer Vision*, pages 75–91. Springer, 2016.

[59] Pedro OO Pinheiro, Ronan Collobert, and Piotr Dollár. Learning to segment object candidates. In *Advances in Neural Information Processing Systems*, pages 1990–1998, 2015.

[60] PySilCam. Pysilcam suite. `https://github.com/emlynjdavies/PySilCam`.

[61] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[62] Frank Rosenblatt. Principles of neurodynamics. perceptrons and the theory of brain mechanisms. Technical report, Cornell Aeronautical Lab Inc Buffalo NY, 1961.

[63] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[64] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

[65] Bryan C Russell, Antonio Torralba, Kevin P Murphy, and William T Freeman. Labelme: a database and web-based tool for image annotation. *International journal of computer vision*, 77(1-3):157–173, 2008.

[66] Aya Saad, Emlyn Davies, and Annette Stahl. Recent advances in visual sensing and machine learning techniques for in-situ plankton-taxa classification. presented at Ocean Sciences Meeting 2020, San Diego, CA, 16-21 Feb., 2020. 636384.

[67] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[68] Heidi M Sosik and Robert J Olson. Automated taxonomic classification of phytoplankton sampled with imaging-in-flow cytometry. *Limnology and Oceanography: Methods*, 5(6):204–216, 2007.

[69] Heidi M Sosik, Emily E Peacock, and Emily F Brownlee. 2014 labeled ifcb images. `https://hdl.handle.net/1912/7350`, 2014.

[70] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[71] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[72] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first AAAI conference on artificial intelligence*, 2017.

[73] Martin Thoma. A survey of semantic segmentation. *arXiv preprint arXiv:1602.06541*, 2016.

[74] TrainingData.io. Trainingdata.io. `https://www.trainingdata.io/`.

[75] Tzutalin. Labelimg git code. `https://github.com/tzutalin/labelImg`, 2015.

[76] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2):154–171, 2013.

[77] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. `https://github.com/facebookresearch/detectron2`, 2019.

[78] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.

[79] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

[80] Haiyong Zheng, Ruchen Wang, Zhibin Yu, Nan Wang, Zhaorui Gu, and Bing Zheng. Automatic plankton image classification combining multiple view features via multiple kernel learning. *BMC bioinformatics*, 18(16):570, 2017.

[81] Hongyuan Zhu, Fanman Meng, Jianfei Cai, and Shijian Lu. Beyond pixels: A comprehensive survey from bottom-up to semantic image segmentation and cosegmentation. *Journal of Visual Communication and Image Representation*, 34: 12–27, 2016.

[82] C Lawrence Zitnick and Piotr Dollár. Edge boxes: Locating object proposals from edges. In *European conference on computer vision*, pages 391–405. Springer, 2014.

Bergum

Instance segmentation of lanktonic organisms using Mask R-CNN for real-time in-situ image processing.

NTNU
Norwegian University of
Science and Technology