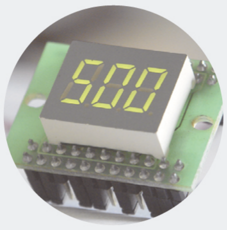
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES



In situ real-time Zooplankton Detection and Classification

PEDRO NUNO DE QUEIRÓS SALCEDAS DE CARVALHO GERALDES

dezembro de 2021

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

In situ real-time Zooplankton Detection and Classification

Pedro Nuno de Queirós Salcedas de Carvalho Geraldes

Master's Degree in Electrical and Computer Engineering

Specialization Area in Autonomous Systems

Supervisor: Alfredo Manuel Oliveira Martins

December 3, 2021

This page was intentionally left blank.

Ao meu pai, de quem nasceu o gosto por este curso, e à minha tia Bela que, como ele, tinha todo o gosto em vê-lo terminado.

This page was intentionally left blank.

Abstract

Zooplankton plays a key-role on Earth's ecosystem, emerging in the oceans and rivers in great quantities and diversity, making it an important and rather common topic on scientific studies. It serves as prey for many large living beings, such as fish and whales, and helps to keep the food chain stabilized by acting not only as prey to other animals but also as a consumer of phytoplankton, the main producers of oxygen on the planet. Zooplankton are also good indicators of environmental changes, such as global warming or rapid fluctuations in carbon dioxide in the atmosphere, since their abundance and existence is dependent on many environmental factors that indicate such changes. Not only is it important to study the numbers of zooplankton in the water masses, but also to know of what different species these numbers are composed of, as different species can provide information of different environmental attributes.

In this thesis a possible solution for the zooplankton *in situ* detection and classification problem in real-time is proposed using a portable deep learning approach based on CNNs (Convolutional Neural Networks) deployed on INESC TEC's MarinEye system. The proposed solution makes use of two different CNNs, one for the detection problem and another for the classification problem, running in MarinEye's plankton imaging system, and portability is guaranteed by the use of the Movidius™ Neural Compute Stick as the deep learning motor in the hardware side. The software was implemented as a ROS node, which guarantees not only portability but facilitates communication between the imaging system and other MarinEye's modules.

Keywords: Zooplankton, MarinEye, object detection, image classification, Deep Learning, Convolutional Neural Networks, Movidius™ Neural Compute Stick.

This page was intentionally left blank.

Resumo

O zooplâncton representa um papel fundamental no ecossistema do planeta, surgindo nos oceanos e rios em grandes quantidades numa elevada diversidade de espécies, sendo um objecto de estudo comum em publicações e artigos produzidos pela comunidade científica. A sua importância vem de entre outros factores do facto de ser a principal fonte de alimento de uma grande parte da vida marinha, desde pequenos peixes a baleias, e de ser um grande consumidor de fitoplâncton, a principal fonte de oxigénio do planeta. O zooplâncton é também um bom indicador de alterações ambientais, como o aquecimento global ou variações rápidas na quantidade de dióxido de carbono na atmosfera, uma vez que a sua abundância depende de diversos factores ambientais relacionados com tais mudanças, sendo não só importante perceber em que quantidades existe nas massas de água do planeta, mas também por que diferentes espécies está distribuído.

Nesta tese é apresentada uma possível solução para a detecção e classificação de zooplâncton *in situ* e em tempo real, recorrendo a uma abordagem facilmente portável de *Deep Learning*, baseada em Redes Neuronais Convolucionais implementado no sistema MarinEye do INESC TEC. A solução proposta faz uso de duas arquitecturas de redes diferentes, uma dedicada à tarefa de detecção do zooplâncton, e outra dedicada à sua classificação, implementadas no módulo de aquisição de imagens de plâncton do sistema MarinEye. A portabilidade e flexibilidade do sistema foi garantida através do uso da Movidius™ *Neural Compute Stick* como motor de *deep learning*, assim como da implementação do software como um nó de ROS, que garante não só a portabilidade do sistema, como também permite uma facilidade de comunicação entre os diferentes módulos do MarinEye.

Palavras-chave: Zooplankton, MarinEye, detecção de objectos, classificação de imagens, *Deep Learning*, Redes Neurais Convolucionais, Movidius™ *Neural Compute Stick*.

This page was intentionally left blank.

Contents

List of Figures

This page was intentionally left blank.

List of Tables

This page was intentionally left blank.

List of Abbreviations

2D	Two Dimensional
3D	Three Dimensional
AI	Artificial Intelligence
ANN	Artificial Neural Networks
API	Application Programming Interface
CCD	Charge Coupled Device
CHDK	Canon Hack Development Kit
CIIMAR	Centro Interdisciplinar de Investigação Marinha e Ambiental
CLAHE	Contrast Limited Adaptive Histogram Equalization
CM	Confusion Matrix
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DL	Deep Learning
DNA	Deoxyribonucleic Acid
DNN	Deep Neural Network
FPGA	Field Programmable Gate Array
FPS	Frames Per Second
GPU	Graphics Processing Unit
HOG	Histogram of Oriented Gradients
INESC TEC	Instituto de Engenharia de Sistemas e Computadores Tecnologia e Ciência
IR	Intermediate Representation
ISEP	Instituto Superior de Engenharia do Porto
ISIIS	In Situ Ichthyoplankton Imaging System
k-NN	k-Nearest Neighbor
LMDB	Lightning Memory-Mapped Database
LOKI	Lightframe On-sight Keyspecies Investigation

LSA	Laboratório de Sistema Autónomos
LVQ	Learning Vector Quantization
mAP	mean Average Precision
ML	Machine Learning
MSE	Mean Squared Error
MSER	Maximally Stable Extremal Region
NCS	Neural Compute Stick
NMDEE	Normalized Multilevel Dominant Eigenvector Estimation
NN	Neural Network
PASCAL	Pattern Analysis, Statistical modelling and Computational Learning
PCA	Principal Component Analysis
PkID	Plankton Identifier
RAM	Random Access Memory
ReLU	Rectified Linear Unit
RF	Random Forest
RGB	Red Green Blue
RNA	Ribonucleic Acid
ROI	Region Of Interest
ROS	Robot Operating System
SBC	Single Board Computer
SDK	Software Development Kit
SGD	Stochastic Gradient Descent
SHAVE	Streaming Hybrid Architecture Vector Engine
SIFT	Scale Invariant Feature Transform
SPC	Scripps Plankton Camera
SSD	Single Shot Multibox Detector
SVM	Support Vector Machine
TEDI	Tese/Dissertação
USB	Universal Serial Bus
VOC	Visual Object Classes
VPR	Video Plankton Recorder
VPU	Vision Processing Unit
WHOI	Woods Hole Oceanographic Institution
ZOOMIE	Zooplankton Multiple Image Exclusion
ZOOVIS	Zooplankton Visualization and Imaging System

Chapter 1

Introduction

Never has the idiom "more than meets the eye" been truer than when referring to Earth's oceans. Because they cover around 71% of Earth's surface and around 90% of it's biosphere, with depths so immense that not even the Sun's light can reach them, and pressures so overwhelming that can easily crush steel submarines, they have been able to hide most of their secrets from humanity. But even if a scuba diver could go for a swim with a flashlight and a magical suit that would resist the pressure, and if he covered the entire 1,335,000,000 km^3 of volume the oceans occupy on Earth, he would still be unaware of many of the secrets the oceans hide, because in the ocean there is a lot more than meets the eye.

One of the most important inhabitants of the oceans, despite its small size, is plankton. The study of these beings is a hot topic in the scientific community, with an ever increasing growth of the state-of-the-art in their detection and classification. This work proposes novel techniques as an attempt to improve the detection and classification of plankton, in an *in situ* real-time scenario.

1.1 Contextualization

Our oceans are abundant with creatures known as plankton. Plankton (from the Greek *planktos*, meaning *wanderer* or *drifter*) are by definition drifting living organisms with limited movement capabilities that inhabit most of Earth's water masses. Their sizes range from less than 0.2 μm to more than 20 cm. The diversity of plankton is well observed in Figure 1.1. Although the term plankton involves an enormous variety of species, they are usually divided in two main groups: phytoplankton and zooplankton.

Phytoplankton consist of bacteria, protists, and mostly of single celled plants. They are primary producers, or autotrophs, meaning that they can produce their own food, without need to consume other living organisms to survive. Although all phytoplankton perform photosynthesis, there are some that get additional energy by consuming other

Figure 1.1: The incredible diversity of plankton. (Christian Sardet / Plankton Chronicles [1])

organisms. Phytoplankton consume carbon dioxide and release oxygen, and scientists estimate that 50% to 80% of Earth's oxygen is produced in the ocean, with the majority coming from plankton. In fact, the smallest photosynthetic organism on Earth, a bacteria called Prochlorococcus, produces up to 20% of the oxygen in our entire biosphere, a bigger percentage than all of the tropical rainforests on land combined [2]. Phytoplankton are also extremely important by making the foundation of the aquatic food web, feeding from the microscopic zooplankton to the gigantic whales, directly or indirectly, and also plays a key role in the global carbon cycle, responsible for regulating the planet's temperature [3]. But despite their importance, there are also dangers associated with phytoplankton. Certain species produce powerful bio-toxins that can kill not only marine life but also people that eat contaminated seafood. Also, when a massive bloom of phytoplankton occurs, and the dead phytoplankton accumulates on the ocean floor, the bacteria that decomposes it consume oxygen faster than it can be replenished, creating entire zones where it is impossible to sustain life, known as dead zones [3].

Zooplankton are the heterotrophic group of plankton, meaning they can't produce their own food, and therefore must consume other living organisms to survive. These creatures play a key role in the oceans ecosystem by serving as the major prey for many large living beings, such as fish and whales, and keep the food chain stabilized by acting not only as prey to other animals but also as a consumer of phytoplankton. Zooplankton are more diverse than phytoplankton and include not only species that spend their whole life cycle as plankton (holoplankton), but also organisms that spend only the initial stage of their life as plankton (meroplankton), as is the case of fish eggs and larvae (icthyoplankton), and crustaceans larvae, like crabs and shrimps. Zooplankton are also good indicators of environmental changes, such as global warming or rapid increases in carbon dioxide in the atmosphere, since their abundance and existence is dependent on many environmental factors, functioning as aquatic "canaries-in-a-cage", as they accumulate over days the effects of hourly changes in water quality [4]. Like so, it's understandable that sudden loss of zooplankton population or a boom of it can result in a great disaster in the ecosystem.

But not only the amount of zooplankton gives valuable information to scientists. Some studies found that the number of some zooplankton species in a water sample is highly correlated to pH. In fact some species of Cladocera and Rotifera increase in abundance when there is a decrease in the water's pH [5], making species richness a better indicator of water acidity than zooplankton density (animals per liter of water), as some more tolerant species can increase in number and replace missing species. Given the facts presented, and the importance of both phytoplankton and zooplankton, it gets obvious why they are a common topic in ecological, biological and overall scientific studies, and why it is important not only to study their numbers in the ocean, but also to know of which different species those numbers are composed of.

1.2 Motivation

Most plankton detection and classification studies make use of sampled cultures and don't consider temporal demands or the behaviour of the beings in their natural habitat, not being developed for real-time or *in situ* applications, existing however systems that aim to study plankton in such conditions, as is the case of INESC TEC's (*Instituto de Engenharia de Sistemas e Computadores, Tecnologia e Ciência*) MarinEye system [6].

MarinEye is an autonomous system for marine integrated physical-chemical and biological monitoring based on contextualized environmental data. This system combines imaging, acoustic, sonar, fraction filtration systems coupled to DNA/RNA preservation and sensors (targeting physical-chemical variables) technologies in a modular, compact system that can be deployed on fixed and mobile platforms. The imaging subsystem includes a plankton imaging camera (targeting both zooplankton and phytoplankton) in conjunction with a coherent illumination strobe, allowing ranging from small water volume and larger plankton sizes imaging to wide environmental images. The MarinEye system is illustrated in Figure 1.2.

In this thesis a real-time *in situ* zooplankton detection and classification system, based on deep learning methods, developed in INESC TEC's autonomous systems laboratory

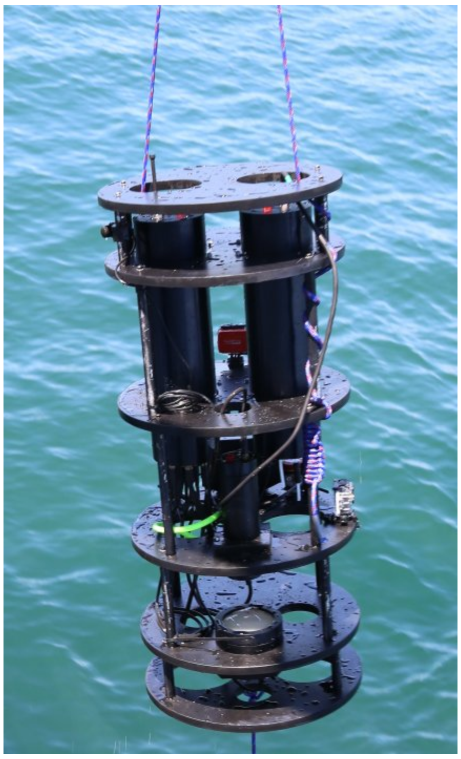


Figure 1.2: MarinEye system with the zooplankton imaging sensor.

(LSA - *Laboratório de Sistemas Autónomos*), is proposed, aiming for a portable solution that can be easily deployed in MarinEye. This project was developed in the scope of the subject Thesis/Dissertation (TEDI) of the Master's degree in Electrical and Computer Engineering, field of Autonomous Systems, at *Instituto Superior de Engenharia do Porto* (ISEP). Throughout this document the term real-time is used in the sense that the methods for detection and classification of plankton occur in the moment the images are obtained, and not in the reactive computing meaning normally used in computer science, that programs must guarantee response within specified time constraints.

1.3 Objectives

The main goal of the project, as mentioned in Section 1.2, is to create an *in situ* zooplankton detection and classification system in real-time, based on deep learning methods, to implement in MarinEye. To achieve this goal the following objectives must be met:

- Study of existing deep learning methods and architectures;
- Collection of a dataset of zooplankton images;
- Apply data augmentation processes on datasets to reduce overfitting and increase the system accuracy;

- Training of detection and classification networks for different datasets;
- Performance evaluation of different networks;
- Integrate system in MarinEye's plankton imaging system;
- Testing and validation of the full system;

1.4 Thesis Structure

In Chapter 1 an introduction on the importance of zooplankton and the zooplankton detection and classification problem is done. The motivation and contextualization for this thesis is also provided, followed by the objectives that this work proposes to satisfy.

In Chapter 2 a review on existing state of the art based on the zooplankton detection and classification problem is done.

In Chapter 3 some fundamentals on deep learning and more specifically on Convolutional Neural Networks (CNN) are described.

In Chapter 4 the proposed system specifications are presented, along with the decision process for the different components chosen. The system's high-level hardware and software architectures are also shown.

In Chapter 5 the methods applied to fulfil the project requirements are explained, as well as the different experiments made to evaluate the decisions taken for the project.

In Chapter 6 the results of the experiments done in the previous chapter are presented and analysed.

Finally in Chapter 7 the overall analysis of the system and the implemented project is done, with a final evaluation of the presented solution and how it satisfied the proposed goals for the project, followed by future work to be implemented for further improvements.

This page was intentionally left blank.

Chapter 2

State Of The Art

In this chapter an overview of the approaches that other work followed in order to solve problems related to this dissertation's subject is presented. This chapter is subdivided into two sections. In the first, a study of other plankton imaging systems is done with focus on how the images are processed in order to detect the plankton and how the classification is done on the detected regions of interest in the image (i.e. areas of the image that show plankton organisms), and if this is achieved *in situ* or even in real-time. In the second section, previous work related to the plankton classification problem that is not necessarily related to any plankton imaging system in specific is presented.

The purpose of this chapter is to acknowledge other plankton imaging systems solutions and possible challenges that must be dealt with to develop MarinEye's plankton detection and classification system, and to innovate the latter so that it can be a novelty in the field, by studying other methods unrelated to any plankton imaging system.

2.1 Plankton Imaging Systems

In this section some plankton imaging systems are presented with focus on their detection and classification methods, particularly if it is done *in situ* or even in real-time.

2.1.1 Lightframe On-sight Keyspecies Investigation

The Lightframe On-sight Keyspecies Investigation (LOKI) [7] system is a towable system that consists on four main units: a plankton concentration net with a mesh size of 200 µm, the main computer unit equipped with a variety of environmental sensors and a Solid State Drive (SSD), the battery unit and the camera system. This system was designed for vertical towing in the water column. In Figure 2.1 is a representation of the LOKI system, with it's main units highlighted.

As is visible in Figure 2.1, the plankton concentration net is attached to the camera unit in such a way that during the vertical towing of the system, the captured plankton

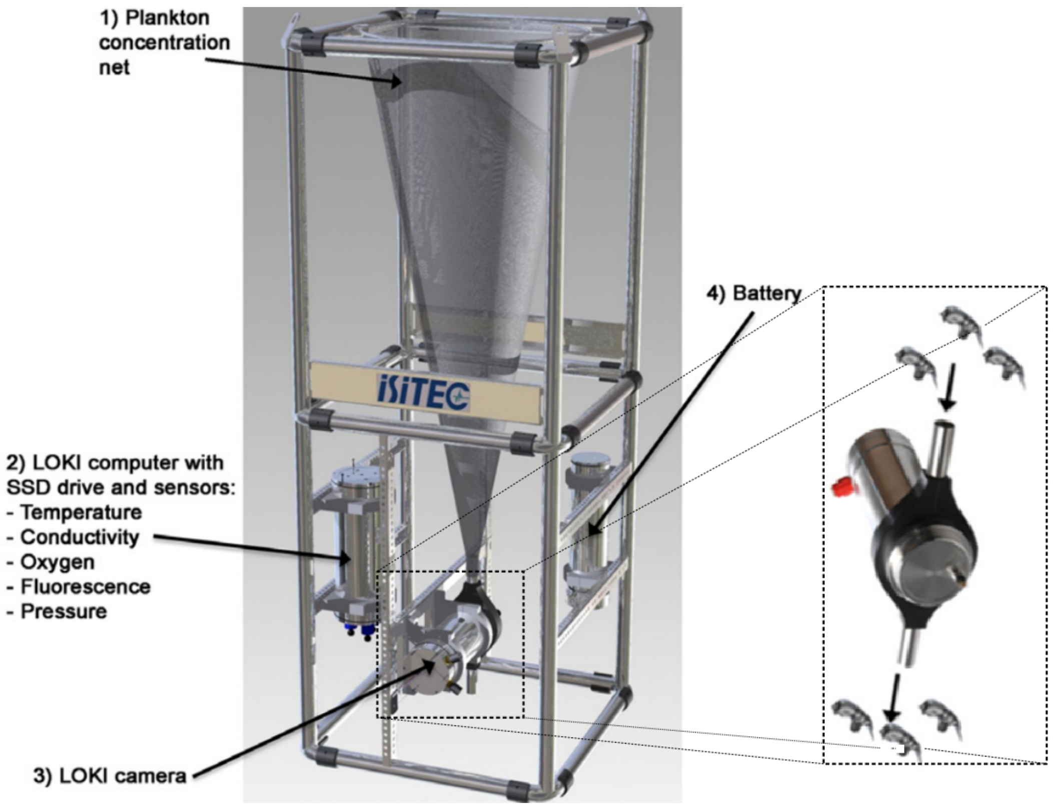


Figure 2.1: The LOKI system and it's main components (adapted from [7]).

organisms enter through the top of the net and flow through a channel that passes in front of the camera. The camera system itself is composed of a Prosilica GC 1380H (Allied Vision Technologies, Canada) monochromatic camera with a Pentax 2514-M lens, and a resolution of 360×1024 pixel at 30 fps (frames per second). For the experiments reported in [7] a dark field illumination and an image resolution of 23 µm per pixel were used. The detection of organisms in each frame is done in real-time, by using a threshold and size filter to detect the regions of interest (ROI). These ROI are cut off from the image with a proportional add-on to width and height of the bounding box of the object's convex hull, and stored on the storage device with a unique time stamp for posterior processing [8]. In Figure 2.2 are visible some ROI captured by the LOKI system. The obtained ROI were later imported into their own software LOKI Browser [8] in order to measure and obtain a set of image parameters, such as object area in mm^2, mean grey value of the pixels or object circularity, used to build the classifier model. A total of 52 parameters were measured. In order to remove images representing the same individuals the ZOOMIE (Zooplankton Multiple Image Exclusion) software [9] was applied to all images, so that the data can be representative of the plankton present in the environment. Image artifacts that could prevent correct zooplankton classification were detected and removed using Adobe's Photoshop CS6. The improved images were later re-imported to LOKI Browser which

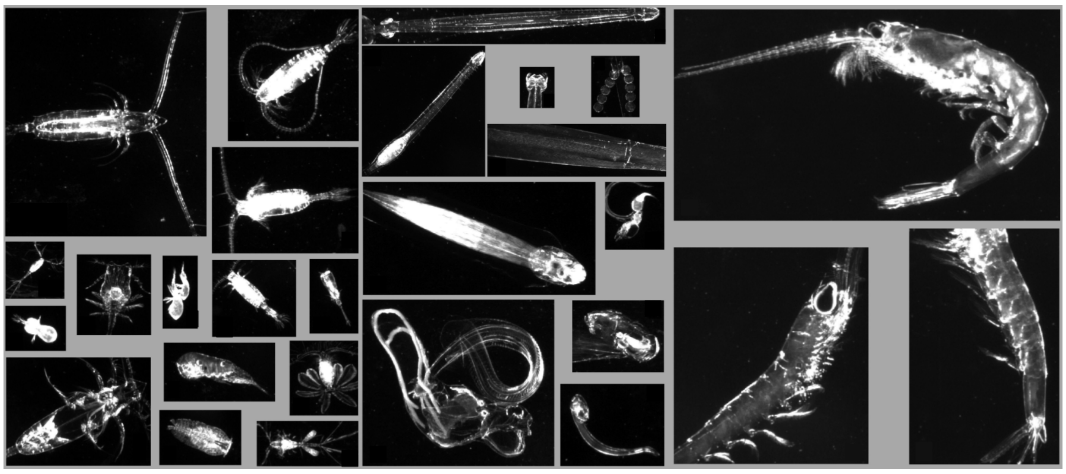


Figure 2.2: Regions of Interest obtained with LOKI (adapted from [8]).

re-measured the image parameters described earlier, to account for changing parameters after removing image artifacts. For automatic classification a Random Forest [10] model was built to distinguish between 114 different categories of zooplankton and other particles using the parameters measured with the LOKI Browser for the nodes on the decision trees of the model. The model used 500 trees with 30 predictors at each node. An overall accuracy and specificity of 86% was obtained. For training and testing of the model a dataset of 14558 manually identified images was randomly divided in two, one consisting of 7252 images for training and the remaining 7306 for testing.

2.1.2 Video Plankton Recorder II

The Video Plankton Recorder II (VPRII) [11] is a distributed system developed for rapid surveys of plankton ranging in size from 100 µm to 1 cm. It consists on two main units: a towable unit with flight control and data acquisition computers, and shipboard computers for supervisory control, data logging and processing, and visualization tools for the user. It is possible to observe the VPRII towable unit in Figure 2.3.

The towable unit contains the plankton imaging system which consists on a 1 MP Pulnix Inc model 1040 camera, that captures 1008×1018 pixel monochromatic images at 30 fps, a manual zoom lens from 12.5 to 75 mm, and a 20 W strobe flashing at 30 Hz, synchronized with the camera shutter. Similarly to the LOKI system this also uses a dark field illumination technique. No image processing or object detection and classification is done on the towable unit. Instead, all video and environmental data from the sensors are sent from the towable unit to the shipboard station via a fiber optic cable, and there the image processing takes place.

The image processing consisted on in-focus object detection, object feature extraction and object feature classification. The following steps are described in more detail in

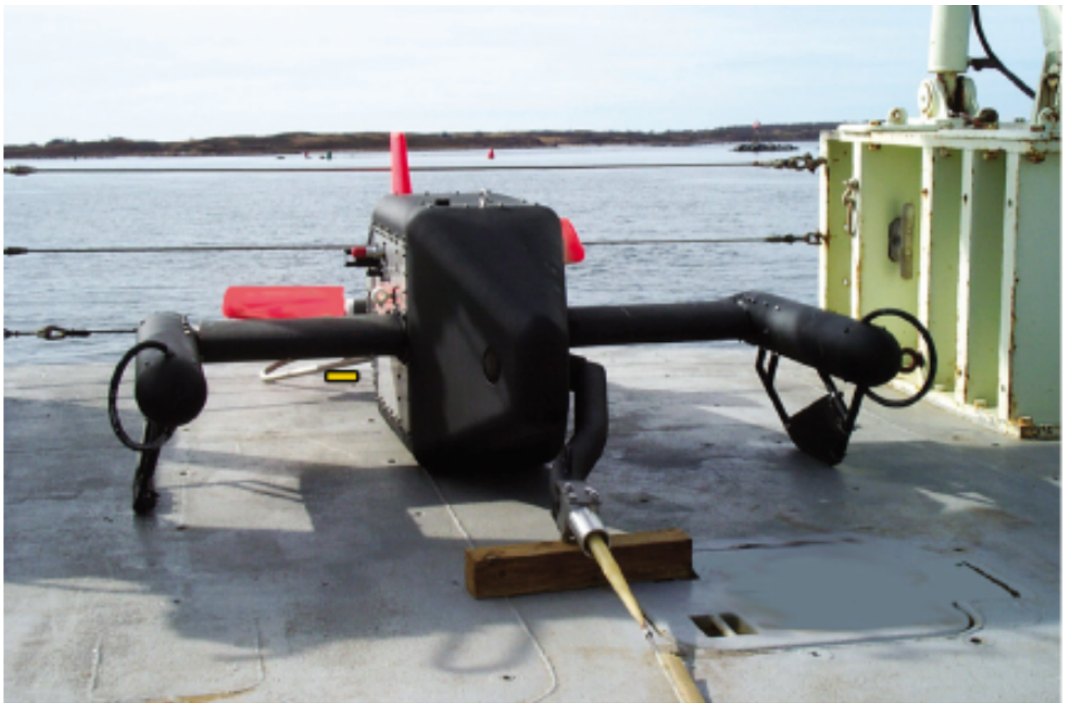


Figure 2.3: The VPRII system [11].

[12]. For the in-focus object detection the ROIs on the image are detected using an edge detection method. The video is corrected for uneven illumination using a running average of 500 frames, followed by a binarization of the images using a user defined brightness threshold. Then the white regions of the image (blobs) are detected and if their size is above a minimum defined value, they are used as a mask on the original image, in order to determine the gradient in grayscale in that ROI, as a measure of the focus level of the object. If it is above a user defined threshold, the ROI of the original image is saved for the next processing steps. From the obtained ROI some feature extraction is applied to use later in the object classification. The collected features include shape-based features (moment invariants, granulometry, roundness, and Fourier descriptors), and texture-based, like co-ocurrence matrices. The features are then combined into a single feature vector with a total of 237 elements. Finally for classification a neural network classifier was trained by manually sorting a set of ROI into different taxonomic groups, extracting and transforming their feature vector, and using the Learning Vector Quantization (LVQ) method [13] to fit neurons to these transformed feature vectors. The resulting classifier model was then used to automatically classify the remaining ROI that were unused in the training moment. The classification method resulted in an accuracy for 7 to 10 classes from 60% to 90% depending on class. In Figure 2.4 is visible a flow chart of the image processing method.

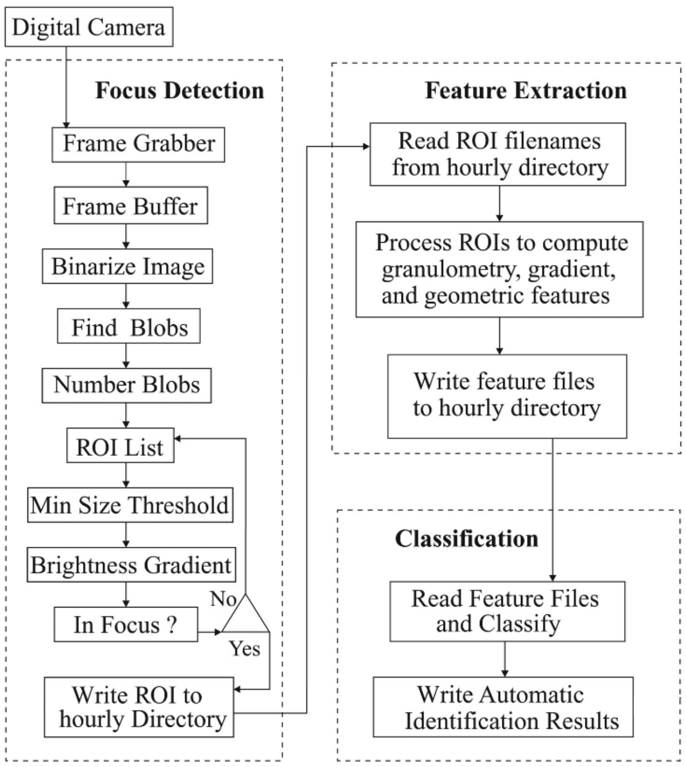

Digital Camera
Focus Detection
Frame Grabber
Frame Buffer
Binarize Image
Find Blobs
Number Blobs
ROI List
Min Size Threshold
Brightness Gradient
In Focus ?
No
Yes
Write ROI to hourly Directory
Feature Extraction
Read ROI filenames from hourly directory
Process ROIs to compute granulometry, gradient, and geometric features
Write feature files to hourly directory
Classification
Read Feature Files and Classify
Write Automatic Identification Results

Figure 2.4: Flowchart of VPRII image processing (adapted from [11]).

2.1.3 Zooplankton Visualization and Imaging System

The Zooplankton Visualization and Imaging System (ZOOVIS) is as the name suggests a system responsible for zooplankton imaging. Visible in Figure 2.5, this system contains two pods seperated by a fixed distance, where the imaging sensor, optics and light source are located. On the illumination pod a specialized optics system enable a higly collimated red beam (625 nm) produced by an LED to be pulsed at 5 ms intervals. This beam travels through the water column separating the two pods and in the camera pod it is focused back to the camera by a set of lenses. The camera itself consists on a high resolution digital camera with a 12 bit, 5.0 MP Charge Coupled Device (CCD) sensor capable of acquiring images at 15 Hz. This setup allows for a long depth of field (30 cm), where objects from 20 to 40 µm and larger can be resolved [14].

Although the ZOOVIS system doesn't perform any image processing in real-time and *in situ*, it just records the images, a study presented in [15] developed a plankton detection and classification method for images obtained with the ZOOVIS system. In their work, Hongsheng Bi et al. developed a semi-automated approach to analyze plankton taxa from images acquired by ZOOVIS within turbid estuarine waters. They proposed a robust procedure to separate objects from non-uniform background in complex noisy images from turbid waters, because the existing procedures were either designed for systems that obtain images from controlled environments with laboratory samples or *in situ* but from waters

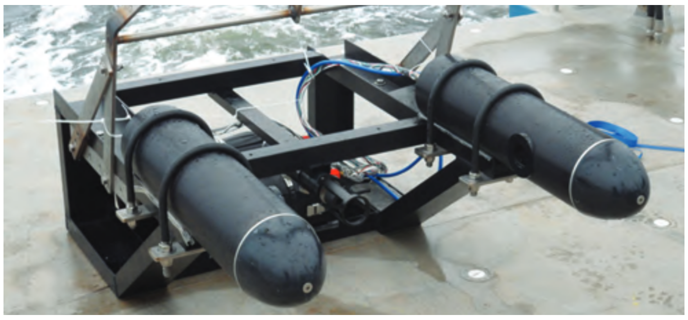


Figure 2.5: The ZOOVIS system [14].

with high clarity. Their process includes four steps: segmenting ROIs, ROI denoising, feature descriptors and taxonomic classification using a Support Vector Machine (SVM).

For the ROI segmentation the method used was considerably different from the ones applied in the previous systems. Global threshold values for binarization, like the methods described previously use, have two main problems in the specific case of plankton identification that occur often, being them:

1. Segmenting large gelatinous zooplankton into separate objects, being complex to merge these parts together;

2. Missing smaller organisms;

Because of these problems, the authors implemented different approaches for different object sizes. For larger objects (> 5000 pixels, at around 0.5 mm^2) they applied Maximally Stable Extremal Regions (MSER). While the basic concept of MSER is similar to thresholding, it differs by selecting only regions which remain nearly the same over a range of thresholds. For the smaller organisms an adaptive threshold was developed. With these two methods, the authors combined the two binary images resultant from each method to obtain one final binary image, from which each detected ROI was then cropped from the original image and saved. In Figure 2.6 the results of this method are seen.

After obtaining the ROIs the next step is to denoise them, since often an individual ROI can contain multiple objects. For this the ROIs are converted to a binary image using the global threshold method. The global threshold method can be used on individual ROI because in these the background tends to be more uniform than on an entire image. With the binarized ROI, the connected components (components that are part of the same organism) are identified. Then the greyscale values of the largest connected component are extracted from the original ROI and the rest of the connected components were assigned with the average gray value. The use of the gray values in favor of the binary values was to retain internal structure, with a special regard to gelatinous zooplankton, which are organisms that given their transparency can provide better information with its texture

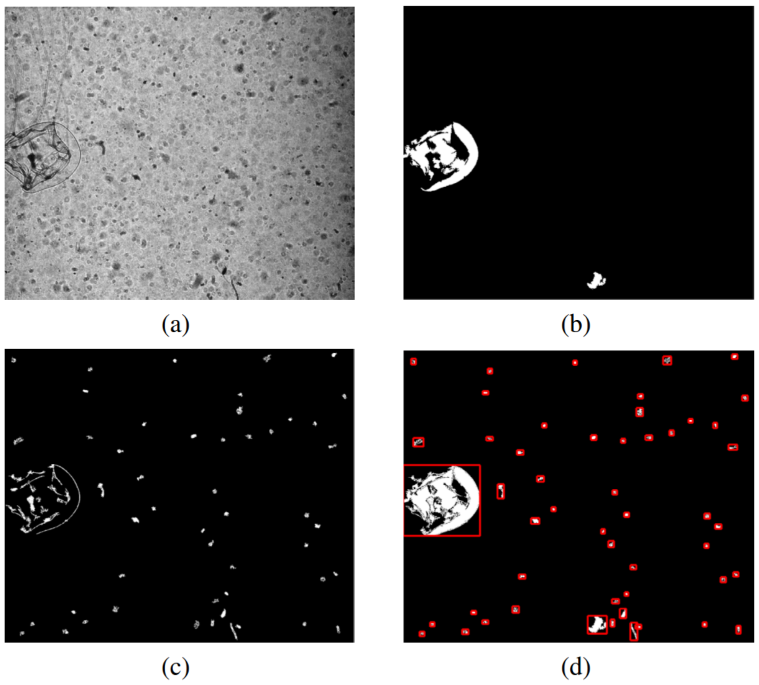


Figure 2.6: Results from the ROI segmentation method, (a) raw image from ZOOVIS, (b) binary image from MSER approach, (c) binary image from adaptive thresholding, (d) combined binary image with the segmented objects highlighted (adapted from [15]).

than with its shape. With the gray values extracted the texture features for each ROI are then constructed.

To classify the denoised ROI into different classes, they are first normalized. Then Histogram of Oriented Gradients (HOG) features are constructed to describe the shape for each ROI. For this each normalized ROI is decomposed into small cells, each cell containing 16×16 pixels. Then each cell is represented by a histogram of edge orientations, with a number of 9 orientation histogram bins. In Figure 2.7 the method of construction of feature descriptors for example ROI is shown.

Classification of the ROI is done in a two step procedure using SVM techniques. In the first step the SVM classifiers were trained using a manually created library composed of three classes: gelatinous zooplankton, arrow-like and copepod-like. For this a library containing 80 arrow-like, 65 copepod-like and 65 gelatinous zooplankton images was used. Since SVM are binary classifiers, a SVM model was created for each of these classes and therefore in this step some ROI could eventually be classified into more than one class. For the second step, each ROI was passed through a group-specific SVM classifier, trained in order to separate target to non-target objects, e.g. for the copepod class the model was trained with a manually constructed library containing 65 ROI of copepods and 985 ROI that did not contain copepods. For the other classes similar classifiers were constructed. Effectively, each model constructed for this second step would function as a

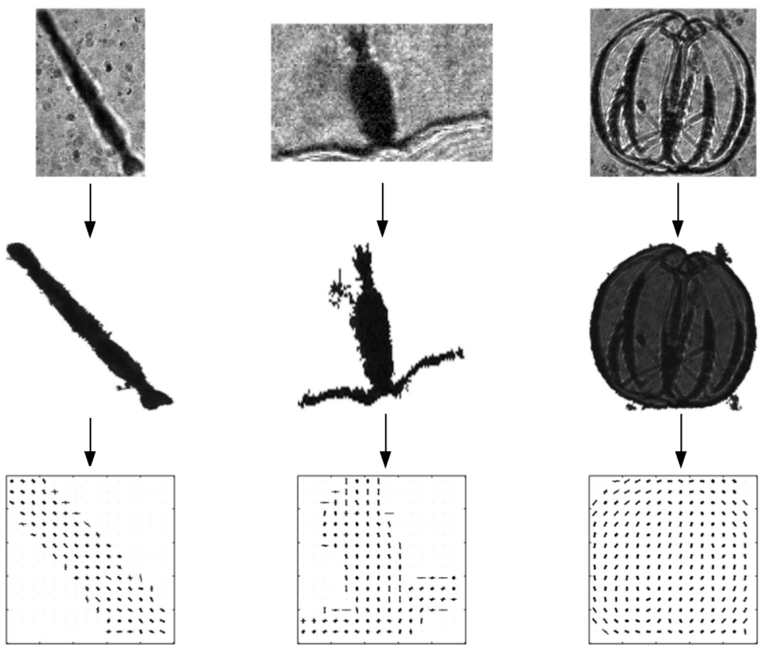


Figure 2.7: Construction of features for example ROI. Top: example ROI. Middle: denoised and normalized ROI. Bottom: HOG feature descriptors. (adapted from [15]).

binary classification model for the previously detected class. The flow of the classification of the ROI can be easily explained as:

1. Classify ROI into gelatinous zooplankton, copepod-like or arrow-like;
2. From the result of the previous classification, classify ROI from an actual member of that class (e.g. copepod) or not an actual member (e.g. copepod-like but not copepod);

The proposed semi-automated method achieved more than 80% of overall true-positive rate for the three classes.

2.1.4 GUARD1

The GUARD1 [16, 17] plankton imaging system was designed for recognition of gelatinous zooplankton in a small, low-cost and easily deployable manner. The development of this system was motivated by the lack of systems designed specifically to detect and measure gelatinous zooplankton quantities, as well as the fact that most of the existing plankton imaging systems mode of operation require towing from a vessel, which is a costly operation. In Figure 2.8 it's possible to see the GUARD1 system in three different enclosures, for different deployment methods, and mission depths.

The systems imaging components consist on a programmable consumer camera Canon GX1 with 12.8 MP and two 1 W LEDs for illumination. The camera's acquisition parameters, such as ISO, exposure time, focal length or iris aperture can be automatically

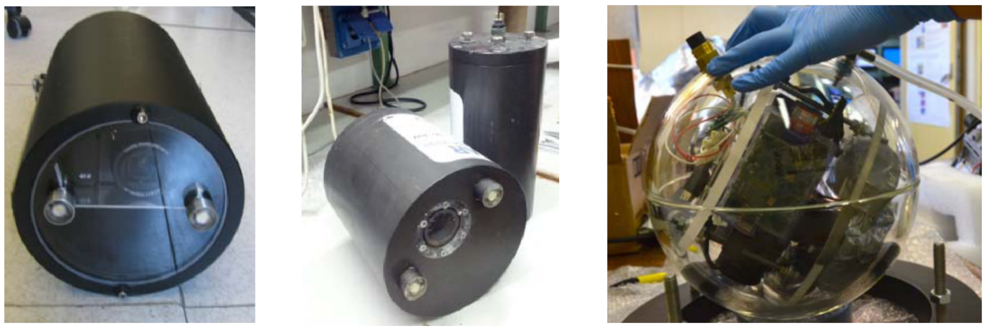


Figure 2.8: Three different enclosures for the GUARD1 system. Left: for deployment on depths up to 40 m. Center: for deployment on depths up to 400 m. Right: for deployment on depths up to 10000 m. [17].

adjusted in order to adapt to the light conditions by using a script based on Canon Hack Development Kit (CHDK) [18] running on the camera firmware. The image acquisition, detection and classification techniques are processed in the system's Central Processing Unit (CPU) board, a Raspberry Pi [19]. The behaviour of the system during the working activities is as follows: the system stays in stand-by mode for a defined time interval; then the light system is activated, if necessary, as well as the image acquisition module, for another defined time interval; when the previous step ends the system returns to stand-by mode. During the image acquisition step, the camera first acquires and stores a predefined number of images. After a number of acquisition sessions, the CPU board system downloads the captured images from the camera storage and runs the detection and classification algorithms on them.

The image detection and classification algorithm consists on five steps: image enhancement, background-foreground segmentation, ROI identification, feature extraction and ROI classification.

Given the transparency and visual characteristics of gelatinous zooplankton, they are often hard to detect on images acquired by a normal camera, given the low-texture and uniform background typical of oceanic environments. This can be observed on the first image of Figure 2.9. Because of this the first step of the GUARD1 image processing consists on image enhancement. For this step the Contrast Limited Adaptive Histogram Equalization (CLAHE) algorithm [20] was used. CLAHE calculates histograms of small adjacent regions of pixels that are equalized separately. The overall equalized image is obtained as a combination of the equalized neighbouring regions.

For the foreground and background segmentation step a simple box-shaped moving average filter with a box area of a size comparable with the size of the expected objects (order of 20 pixels) was used. This filter transforms the original image in a binary image, separating the image background and foreground. The foreground image regions, or blobs, are then defined as the set of pixels with higher intensity values than the background and

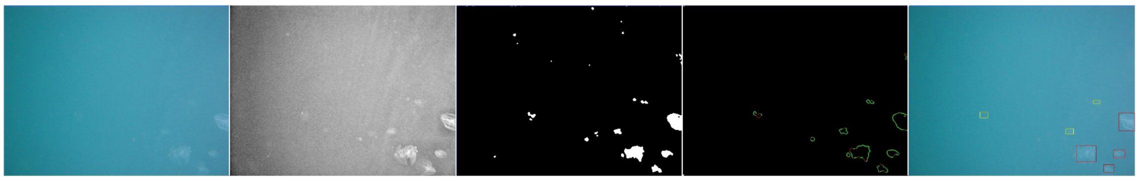


Figure 2.9: GUARD1 image processing steps. From left to right: original image, image enhancement with CLAHE, background/foreground segmentation, contour extraction from original image with Sobel operator, and ROI identification [16].

exceeding a global threshold value. To tune the segmenting filter, information about the radiometric nature of the jellies is used, as they appear brighter than the surroundings when illuminated.

After segmenting the image in foreground and background the next step is to select from the detected blobs which are candidates to be objects of interest. In fact in underwater images it's common to have artifacts such as light reflections, bubbles or other suspended particles that will be identified as foreground in the previous step but must be avoided in the identification step. Since these artifacts are usually characterized by a blurred contour, they can be easily identified by analysing the internal/external contour gradient. For this gradient analysis the authors used a filtering process based on the Sobel operator [21] applied on the original image. Then the contours obtained by this operation are compared with the blobs extracted in the binary image and if the number of pixels of the Sobel contour is less than 50% of the pixels of the morphological contour of the blobs on the binary image, then the blob is not considered relevant. In Figure 2.9 the image processing steps are shown.

Finished the steps for obtaining the image ROI, it's now necessary to process them for classification. For this, a set of features are extracted from them to create a feature vector. The extracted features can be divided in two groups, shape based and texture based, similarly to the VPRII method. On the shape based group a set of seven features were obtained, being them: the lengths of the minor semi-axis, minor and major axis of the ROI oriented bounding box, describing the size of the relevant subject; the eccentricity, being a measure of how the ROI differs from a circle; the solidity, being the ration between the area of the ROI and the area of the corresponding convex hull; the area and the perimeter of the ROI. On the texture based features a set of four were extracted: the histogram shape index, which is obtained by transforming the ROI in a grey scale image and extracting a histogram h of the pixel intensities, and is defined as the standard deviation of h after the histogram normalization; the standard deviation of the mean grey level, which captures the variation of the pixel intensity with respect to the ROI mean grey value; the entropy of h; the normalized contrast index, defined as the ratio between the difference in the mean grey value inside the ROI and outside the ROI but within the bounding box, and the mean grey value inside the whole bounding box.

For the binary classification (gelatinous zooplankton or non gelatinous zooplankton) of the ROI using the extracted features three methods were tested: Elastic Net based on Tikhonov regularization (TR) [22], SVM and Genetic Programming (GP) [23]. In these methods the authors were not only interested in acquiring overall accuracy, but also which of the extracted features were essential for the algorithm to provide such performances. In their results there were no significant differences in the performance of the three methods in terms of prediction accuracy and other performance indicators, such as true positive rate, false positive rate and false negative rate, however the authors decided to implement the Genetic Programming approach in GUARD1 since this method could achieve similar accuracy to the other methods while using less features than the others. In fact, both the Tikhonov regularization and the SVM used eight features from the eleven features extracted, while the Genetic Programming used only two, being them the length of the minor semi-axis and the eccentricity. More details about the classification methods tested by the authors can be obtained in [16, 24].

2.1.5 Scripps Plankton Camera

The Scripps Plankton Camera (SPC) system [25] was developed by the Scripps Institute of Oceanography for *in situ* real-time plankton observation. The system consists on three distinct nodes: the imaging system node, a server to manage data and to manage analysis and an interface for remote clients to observe and annotate images. Depending on the target size of plankton to obtain images from, four different setups have been developed, the MACRO-SPC, the MINI-SPC, the MICRO-SPC and DUAL-SPC, the latter consisting on the imaging components of the MINI- and MACRO- systems in the same housing. The MACRO-SPC used a projection lens array illumination technique while the others used dark field illumination. In Figure 2.10 the high level system architecture is seen, alongside the actual system.

The ROI detection occurs on-board in real-time and consists on down-sampling the raw image by averaging pixels into 4- or 16-pixel blocks. Then a Canny edge detector is applied to detect edges on the image, followed by a region filling algorithm used to fill the contours. Then bounding boxes are drawn from the centroid of the detected blob and its dimensions are doubled from the blob's width and height as to completely enclose the detected objects. Then the ROI are cropped from the original image and stored locally before being exported to the remote network storage. Since on a first iteration of the SPC the processing unit consisted on a 1.8 GHz Quad Core Odroid XU3 board with 2 GB of RAM this process was done at around 8 frames per second, but the classification models had to be run remotely, given their computational demands. On newer iterations of the SPC the system used a NVIDIA Jetson TX1 embedded GPU, which would allow real-

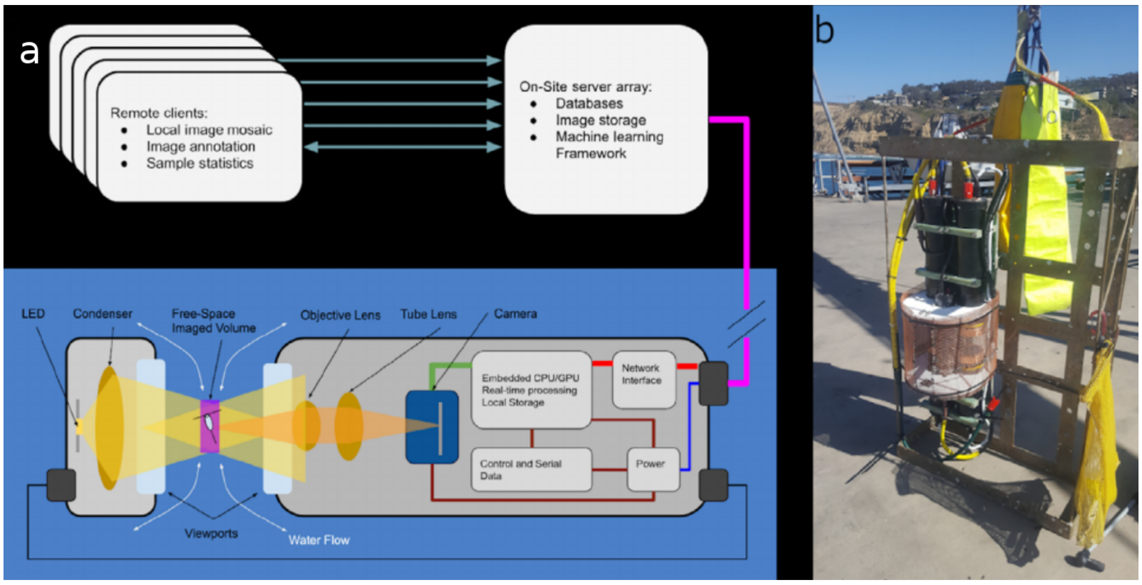


Figure 2.10: The SPC system. (a) High-level system architecture. (b) MINI- and MICRO-SPC systems before deployment [25].

time operation at 20 frames per second, as well as simultaneous video and image capture, possibility of running multiple cameras and on-board classification using complex models, such as deep neural networks. With this new processing unit *in situ* classification would be possible, being dependent only on the creation of a dataset for the desired purposes on each specific case, and training the chosen models.

2.1.6 Discussion on Plankton Imaging Systems

Here a discussion of the main issues with the presented plankton imaging systems is done, considering the approaches of each method for the detection and classification of plankton.

The LOKI system is capable of *in situ* plankton detection using classic image processing methods. However the method to obtain the ROI is dependent on manually chosen threshold values, which with non optimal settings may be missing the detection of some individuals. The detection method applied will also detect and save to the system's storage images of any object that is detected, be it organic or inorganic objects of no interest, along with the problems of global thresholding already specified in Subsection 2.1.3. The way that the system was designed also means that organisms bigger than the plankton net mesh size will not be catalogued. The classification is not performed *in situ* and is too dependent on manual intervention. First the detected ROI require some image enhancement techniques in order to provide relevant features, and the use of many extra software tools is needed, such as the LOKI Browser, the ZOOMIE and Adobe's Photoshop. Then the choice of parameters to measure and to create the classification model must also be manually decided, which means that model accuracy is dependent on manual choice of parameters to measure. The authors claim that the chosen parameters in [7] provided good results, which is true, but if one was to try to replicate the results for other images, datasets or even different classes of zooplankton, some knowledge, on both programming and zooplankton characteristics, and experimentation is required, and a poor choice of parameters has a high impact on overall system results. Finally there is the need to choose a good set of decision trees to create the Random Forest based on the measured parameters.

While the VPRII system's plankton detection and classification method is technically made *in situ*, it is done in a distributed manner, since the unit responsible for image acquisition is not the same as the one responsible for actually processing it. This means that the system performance is dependent on the reliability of the communication channel, in this case on the fiber optic cable. The whole system is also not exactly portable, since the entire system consists on the towable unit, sheave, cable and two computers on board of the ship. The towable unit alone weights 400 kg and measures 2.6 $\times$ 2.0 $\times$ 0.6 m (L $\times$ W $\times$ H), and shipboard setup takes a few hours by trained personnel. As is the case with the LOKI system, VPRII images are also monochromatic, and so colour information can not be used for plankton identification. The detection of the ROI also depends on many manually selected values, such as for brightness, focus and object size, and relying on classic image processing methods, it suffers from some of the same problems as LOKI, such as detecting any visible object, being it plankton or any non organic objects. The classification method consisting on machine learning methods like neural networks and LVQ also require some expertise to create a good model, as well as many feature extraction methods, and so can be quite complex.

The ZOOVIS system doesn't actually perform plankton detection and classification *in situ* or even in real-time, however there was a method developed to process the images obtained by the system. In this method the problem of noisy *in situ* images was addressed, as well as the problem of global threshold values for image binarization. For the detection of the ROI this method is indeed an improvement on the two previously mentioned, and the classification method has a somewhat acceptable result. However the main downside to this method is the low number of classes that the system can classify the ROI into, and improving the system to classify for other classes meant retraining the first step of the classification, and then creating a new SVM model for binary classification for each new class. As the number of classes increases, so do the computation demands and the false positive rate.

The GUARD1 system addressed the problem of detecting gelatinous zooplankton in images obtained *in situ* and provided good improvements in this aspect. The system is also very small and easily deployable while still capable of performing detection and classification *in situ* of the chosen organisms. However, there are flaws in the method of image acquisition. Since it captures and stores images in the camera memory based simply on time defined variables, processing them afterwards, it can easily store high amounts of images with no actual value, i.e. with no objects of interest in frame, and therefore waste resources with unimportant data. The classification method still depends on a decided set of features obtained in a variety of methods, and problems previously mentioned in the other systems related to this are also true in this. With the chosen classification method of Genetic Programming it was shown that this was not such a big issue for the specific problem of gelatinous zooplankton, but complexity can easily increase if one was to try to use these methods for a non-binary classification. For such a case, the Genetic Programming approach can be non satisfactory, since the search space for a fit individual increases considerably.

The SPC acquires high resolution coloured images of plankton of various sizes, and has a computing system capable of performing *in situ* real-time detection and classification, although classification is not applied in the system as default. However it is only a matter of reprogramming and applying the designed and trained desired models, as the system is capable of running even computational expensive algorithms such as deep convolutional neural networks. The main drawback is the cost of such a processing unit, and the power it may consume.

2.2 Plankton Classification Approaches

The success of machine learning methods in the field of pattern recognition made way for the growing use of such methods in complex detection and classification scenarios, being promising in the solution of the zooplankton classification problem. The study of image classification started a few years ago, with the development of the feature designed methods like Learning Vector Quantization (LVQ), k-Nearest Neighbor (k-NN), Random Forest (RF), Support Vector Machine (SVM), etc, and in the recent years with the advances of deep learning, the Convolutional Neural Networks (CNN).

There are some efforts to devote to the study of zooplankton enumeration and classification methods, which mostly consist in manual counting and identification of these beings, proven to be an extremely time consuming and hence costly task. To overcome this problem, with the evolution of machine learning algorithms, a set of automated plankton analysis approaches arise.

The work of Philippe Grosjean et al. [26] focuses on identifying, counting and measuring digitalized zooplankton samples using the ZooScan system [27] which permits rapid and complete analysis of preserved zooplankton samples and stores the data in digital form. This method consists on image processing on sub-sampled images and using several methods of machine learning for classification, such as k-Nearest Neighbours, Learning Vector Quantization, decision tree and recursive partitioning methods, Support Vector Machine and Random Forest. They also tested methods in which two or more differ-

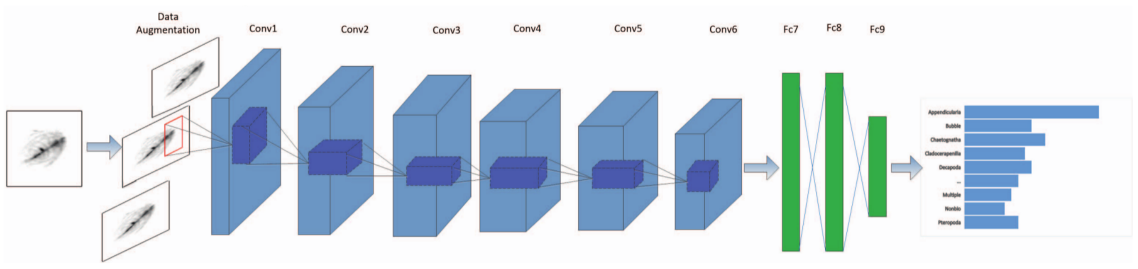


Figure 2.11: ZooplanktoNet architecture [28].

ent algorithms are combined. In their experiments, Random Forest and Support Vector Machine provide more satisfying than other algorithms in zooplankton classification. To attain the best results this system requires a large number of features to be used in the selected learning algorithms, it must not be applied to species-rich collections for the most difficult specimens in the samples human intervention is used.

In the work of Gaby Gorsky et al. [27], focusing on image analysis for zooplankton classification, a semi-automatic approach is proposed, which entails an automated classification of images and a manual validation, allowing a rapid and accurate classification not only of zooplankton but also abiotic objects. For the project the ZooScan system was used along with ZooProcess and Plankton Identifier (PkID) software. For the extraction of ROIs in the scanned images, ZooProcess's image analysis method consists in scanning and processing a blank (background) image followed by the scan of the sample. The raw image of the sample is normalized in order to subtract the blank image from it and the extraction and measurement of the objects (ROI) is made. For plankton identification with PkID a learning set is built and the construction of the classifier is made and applied to predict the I.D. of unidentified objects, finally followed by manual validation of the results. For the classifier construction step a comparison between a few classifiers available in PkID was made (k-Nearest Neighbor, SVM, Random Forest, a decision tree algorithm and Multilayer Perceptron), with Random Forest providing the best results. The confusion matrix (CM) of the learning set provided for most categories a recall (rate of true positives) of about 80% and a contamination rate (false positives) smaller than 20%, and so stated as not being accurate enough for ecological studies.

Jialun Dai et al. proposes two different convolutional networks in their papers [28] [29]. The first, ZooplanktoNet, aims to classify zooplankton automatically and effectively. The deep network, strongly inspired on AlexNet [30] and VGGNet [31], is characterized by capturing more general and representative features than previous feature extraction algorithms. The ZooplanktoNet architecture is visible in Figure 2.11. Data augmentation, consisting in image transformations like rotation and translation on the training images of the used dataset, is incorporated in order to reduce overfitting for lacking of zooplakton images. The dataset consisted of microscopic and grayscale images captured by ZooScan

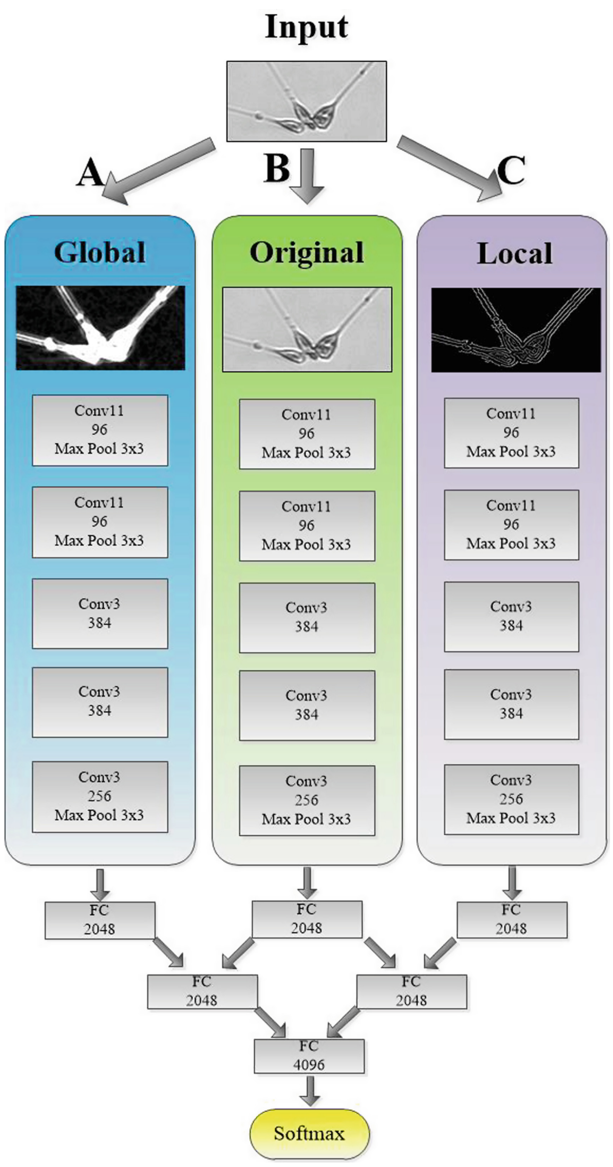


Figure 2.12: Hybrid architecture proposed by Jialun Dai et al. in [28].

system and involved 13 classes with 9460 images. With their experiments they concluded that a ZooplanktoNet with 11 layers achieved the best performance with a final accuracy of about 93.7%.

For the second, a hybrid convolutional neural network for plankton classification, inter-class similarity (similarity between planktons of different classes) and intra-class variance (differences in organisms of the same class) is taken into account and so two feature extraction methods are proposed, one to obtain object global features (shape and setae), and another, mainly based on canny edge detector, to obtain local feature (zooplankton texture). The proposed network architecture consists of three sub Alex networks in a pyramid fully connected structure, one for training on global feature images, a second

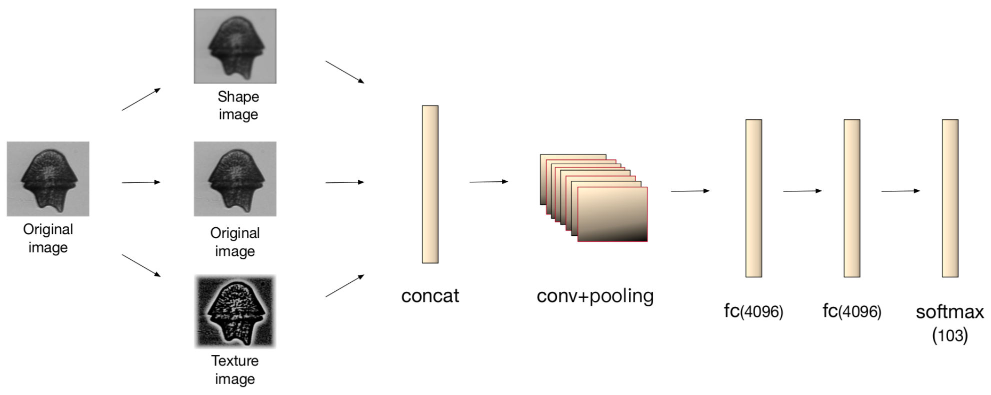


Figure 2.13: Hybrid architecture proposed by Jinna Cui et al. in [32].

for training on the original images, and the third one for training on the local feature images, merging the different inner products from each sub networks in the end. This network achieves a accuracy rate of 95.83% on a 30 plankton classes dataset. However, it's stated that the distribution of plankton in their dataset is highly inhomogeneous, leaving classification of the plankton effectively on an unbalanced data set for future work. In Figure 2.12 the proposed architecture for this hybrid method is shown.

Jinna Cui et al. [32] propose a solution consisting of a feature extraction method followed by a hybrid convolutional neural network for plankton classification. The Woods Hole Oceanographic Institution (WHOI) Plankton dataset [33], which contains about 3.6 million *in situ* images labeled into 103 classes, was used. A new texture feature extraction method based on Gaussian filter is proposed, with the Gaussian high-pass filtering allowing to get textures of plankton which can not be identified clearly in the original images. Since the Gaussian high-pass filtering leaves the image too dark, some image enhancement is applied after the filter. Texture and shape features are emphasized in the hybrid CNN, claiming to improve classification accuracy with their method. For the shape feature extraction a Gaussian low-pass filter is used, while for the texture feature extraction a Gaussian high pass filter. For the image enhancement a logarithmic image enhancement method was adopted. The hybrid CNN is based on AlexNet and consists of adding a *concat* layer before the first convolutional layer, which concatenates the three image inputs (original image, texture image and shape image) to the convolutional layer. The results for a 30 classes dataset show improvements in accuracy comparatively to Dai's hybrid method, and also shows better accuracy with the three inputs (96.58 %), then to two (94.93 % with original and shape and 95.27 % with texture and shape and 96.10 % with original and texture) or one (94.75 %). When trained for the whole WHOI dataset, with 103 classes, where images from 2006-2013 database were used as train dataset and images from 2014 were used as test, an accuracy of 94.32 % was achieved. In Figure 2.13 the proposed architecture is shown.

From the methods presented in this section it is seen that the ones based on CNNs show multiple advantages: first, these methods are consistently shown to achieve higher accuracy values on the datasets they were trained with; they also seem to be able to manage datasets with a larger number of classes; they don't require previous feature extraction, so there is no need for specialized knowledge in plankton from the user, or prior setting of parameters. For these reasons it seems that CNNs are the best candidates for the final solution to implement on the proposed work.

Chapter 3

Fundamentals

In the previous chapter it was clear that the latest advances in the plankton classification methods are resorting to something called Convolutional Neural Networks, or CNN, replacing older methods such as SVM, k-NN or Random Forest, a scenario common to most image classification tasks. In fact in the last decade CNNs have been dominating the state-of-the-art on image classification, and are, at the time of this writing, the most successful at it. But what exactly are Convolutional Neural Networks? To answer this it's first necessary to explain other concepts such as Deep and Machine Learning, or what is *neural* about "Convolutional Neural Networks".

3.1 Learning in Artificial Intelligence

Artificial Intelligence, or AI, has been for decades of great interest in various topics, from technology to science-fiction novels and movies. It can be described as the study of methods and devices that can perceive their environment and act on it in order to achieve a given goal, without human intervention. As Pedro Domingos writes in *The Master Algorithm* [34], "[T]he goal of AI is to teach computers to do what humans currently do better, and learning is arguably the most important of those things". Enter Machine Learning.

Machine Learning (ML) is a subset of AI where the machine learns how to complete a given task without being explicitly programmed on how to do it, by being fed lots of training data and generating a good model to predict the correct values for new similar data. A frequently used definition for Machine Learning is provided by Tom Mitchell [35]: "A computer program is said to **learn** from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E". With this definition, in the plankton image classification problem the task T would be recognizing and classifying plankton organisms within images, the performance measure P would be the percentage of images correctly classified, and the training experience E would be a database of plankton images with their correct classification labels.

Machine Learning problems are usually divided into two broader classifications: supervised learning and unsupervised learning. In unsupervised learning the problem is approached with little to no knowledge on what the results should be, as there are no labels for the data. Instead, unsupervised learning algorithms try to derive relationships in the data based on their variables, clustering it in groups when such relationships or patterns are discovered by the algorithm. A popular example of unsupervised learning is the k-means algorithm.

In supervised learning a dataset is given to the learning algorithm together with labels that show what the correct output should be for the given data, like the example previously mentioned of plankton images and their labels. As examples of supervised learning algorithms there are the ones presented in the introduction of this chapter, such as SVM, k-NN, Random Forests and also the aforementioned Neural Networks, or more specifically, Artificial Neural Networks (ANN).

3.1.1 Neurons and Layers

If one wanted to classify a random picture of a zooplankton organism as being a copepod of the order calanoida, for example, he could look for a particular set of features that might help, such as:

- Size;
- Width;
- Length;
- Eccentricity of the body;
- Number of antennae;

and by analysing the quantitative values on these features he could decide if it is indeed an image of a copepod or not (in fact this is, in a very basic form, the idea behind the binary classification methods of some plankton imaging systems presented in Chapter 2). Because these variables weight in differently when defining a copepod (e.g. a calanoida copepod may vary greatly in size but will always have two pairs of antennae), one way to make a decision is to apply different weights to these variables, multiply each variable with their weight, sum the result of each multiplication, and pass the result through a function, called activation function, that converts it into a "yes" or a "no", as seen in Figure 3.1.

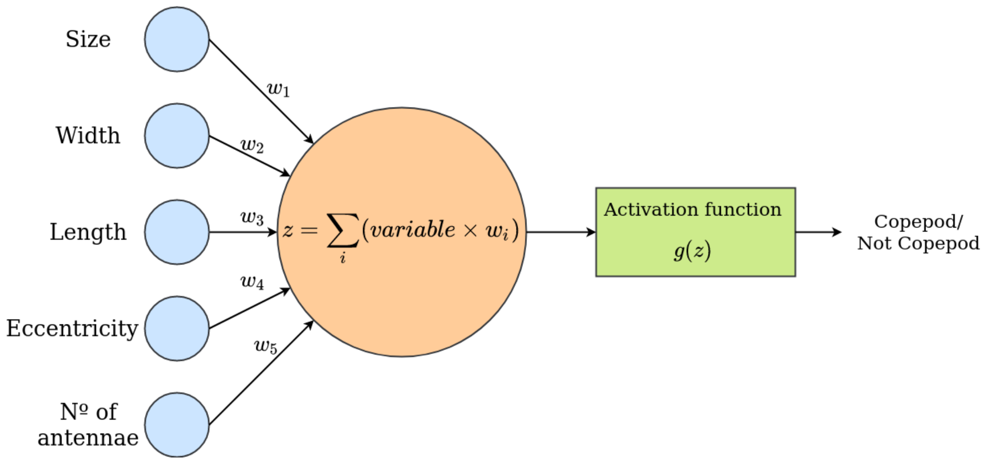


Figure 3.1: Possible classification model for a Copepod.

The model in Figure 3.1 can be generalized for a variable number of inputs a, and use the Heaviside step function (equation 3.1) as the activation function:

$$g(z) = \begin{cases} 0, & \text{if } z < 0 \\ 1, & \text{if } z \geq 0 \end{cases} \tag{3.1}$$

It is also possible to add a bias value b before the activation function to change how easily the output becomes active. If these modifications are applied to Figure 3.1 what is obtained is what is called a Perceptron (Figure 3.2) created by Frank Rosenblatt in 1958 [36], a predecessor of Neural Networks.

Sometimes it is necessary to obtain other non-binary values, instead of a ”yes” or ”no” output, for instance to represent a probability of the result. For this it is necessary to use different activation functions, with the most commonly used being the ReLU (**Re**ctified **L**inear **U**nit), the Sigmoid function, or the Hyperbolic Tangent, visible in Figure 3.3.

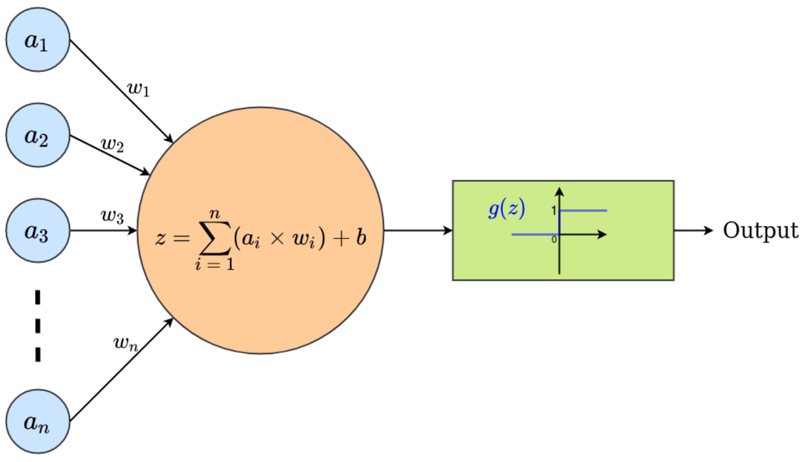


Figure 3.2: The Perceptron.

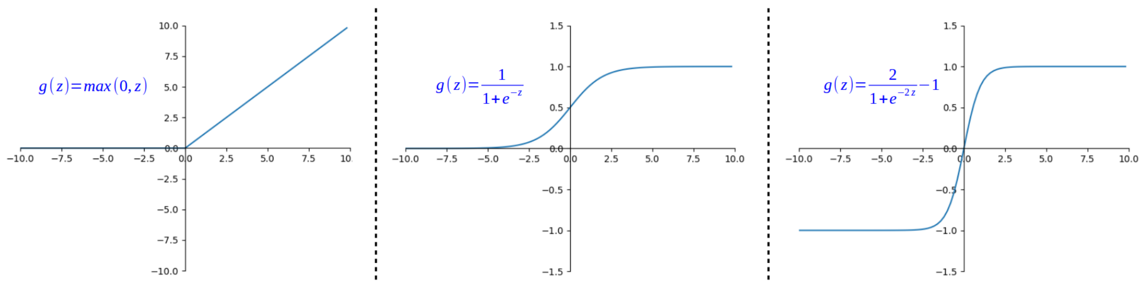


Figure 3.3: Activation functions commonly used in Neural Networks. Left: ReLU. Center: Sigmoid. Right: Hyperbolic Tangent.

A unit that consists of the sum of the product of multiple inputs and their weights, coupled with a bias value and an activation function that outputs only one value will from now on be referred to as a neuron, and the value it outputs is called its activation. In fact, the Perceptron, as well as their successors the Neural Networks, were indeed inspired by the way real neurons work in the brain. Neurons receive multiple signals through special extensions of the nerve cell called dendrites, and from these signals it decides whether or not to emit a signal itself forward to other neurons. In case the neuron decides to transmit the signal forward, it is done so through an extension of the cell's body called axon. Such similarities between biological and artificial neurons are better evidenced in Figure 3.4. However, it is worth noting that the Perceptron and Neural Networks are only *inspired* by, and do not actually work like real neurons.

If several neurons are stacked together at the same level (i.e. the inputs are the same) what is obtained is called a layer. By adding multiple layers in front of each other the result is a Neural Network. Every Neural Network has an input layer, that receives the input data and passes it forward (its neurons activation is the value of the received data), an output layer, that outputs the prediction results, and the hidden layers that stand between the previous two, and depending on their inputs, different neurons will be activated, transmitting information to forward layers that act the same way. In Figure 3.5 is an example of a Neural Network. It is important to note that each neuron will have

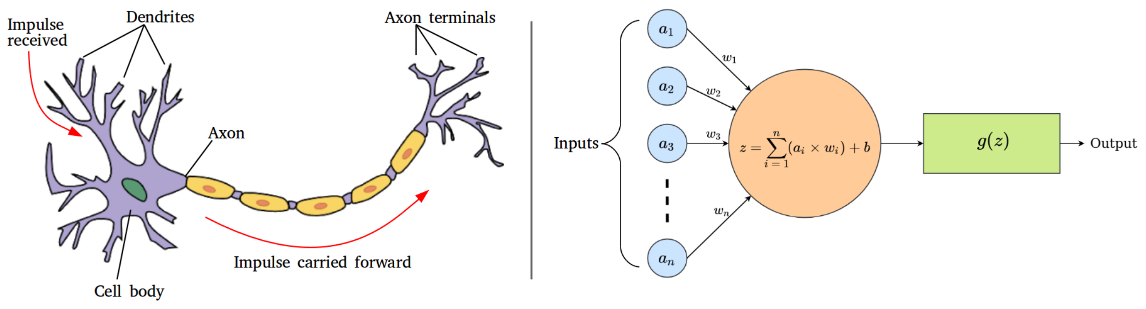


Figure 3.4: Comparison between a biological neuron and an artificial neuron.

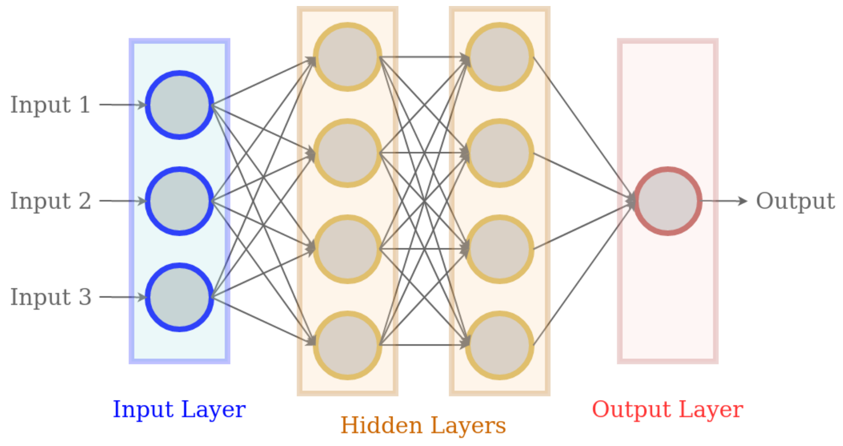


Figure 3.5: An example Neural Network with two hidden layers. Each circle is a neuron with its own weights and bias values, and activation function, except for the input layer, where the neuron's activations are the input data value.

its own values of bias and input weights, otherwise every neuron in the same layer would become active with the same inputs.

In the case of image classification the input data will be an image, and the output can be the probability for the given image to belong to a specific class. In this case the number of neurons in the input layer is determined by the number of pixels in the image, and the activation value of each neuron will be the value of the corresponding pixel. The number of neurons on the output layer will be the number of classes that the image can be classified into, and their activation values will be the probability for the image to belong to the corresponding class. In Figure 3.6 is an example for such a network for zooplankton classification.

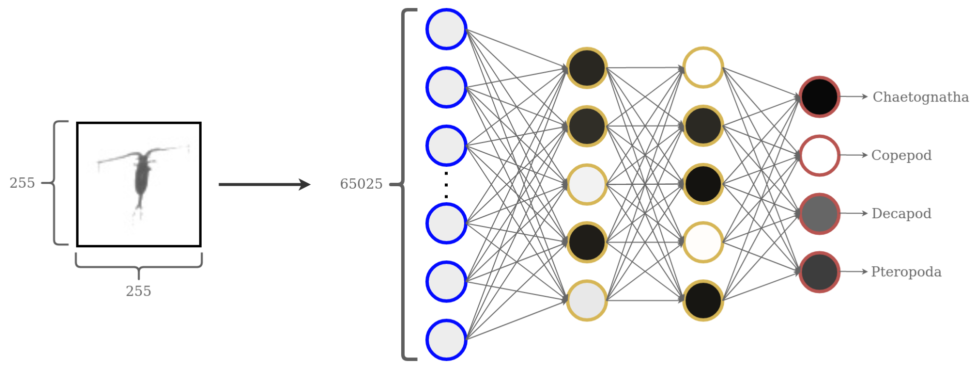


Figure 3.6: Example of a Neural Network for zooplankton image classification.

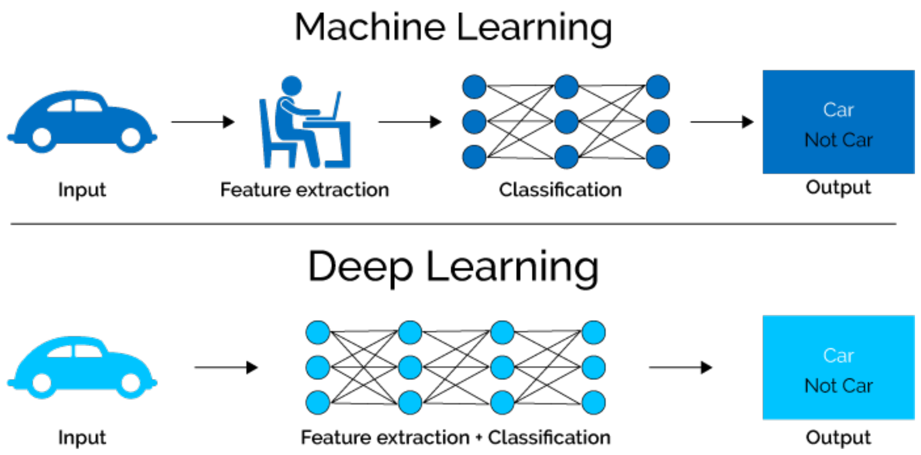


Figure 3.7: Difference between Machine Learning and Deep Learning in the feature extraction process (from [37]).

3.1.2 Deep Learning

The number of hidden layers is what determines the depth of the network. If the number of hidden layers is greater than two then the network is a Deep Neural Network, and the more layers it has, the deeper it is. Deep Neural Networks are the main components of Deep Learning, while Deep Learning itself is a subset of Machine Learning, the same way that Machine Learning is a subset of Artificial Intelligence.

In Deep Neral Networks (DNN), each hidden layer is responsible for picking up features on the output of the previous layer, and the deeper the network goes, the more complex and abstract data becomes, creating a hierarchy of low-level to high-level features. This allows the Deep Learning algorithm to solve higher complexity problems with non-linear data.

Two advantages of Deep Learning over Machine Learning algorithms is that DL performance tends to keep improving with the amount of training data, while ML models reach a saturation point where performance does not increase with increasing training data. The other advantage of DL is that feature choice and extraction is not done manually, as the network is responsible not only for classification but also for feature extraction (Figure 3.7).

3.2 How does a Neural Network learn?

When a non trained Neural Network is fed with some input data, e.g. an image of a copepod, the output will be far from the desired, because the parameters that take place in the decision process, the weights and the biases, will have random values. If the values of the weights and biases are random, so will the active neurons in the hidden layers be, and therefore the neurons on the output layer. So what is needed for a Neural Network (NN) to learn is an algorithm that feeds the network a lot of training data, which for image classification networks consists on images and their class label, and readjusts the weights and biases based on the classification results, in order to improve the performance on that same training data. But before specifying what this algorithm is or does, it is first necessary to understand a few other concepts.

3.2.1 Cost Function

Picking up the example network from Figure 3.6, the output layer can be seen as a vector $\vec{v}$ where each position of the vector is the activation of each output neuron, or the probability for the given image to belong to each corresponding class, and $\vec{v_d}$ is a vector which contains the values for the desired output. If the network is not well trained then the vectors could be something like Equation 3.2 for a given image of a copepod:

$$\vec{v} = \begin{bmatrix} 0.89 \\ 0.21 \\ 0.07 \\ 0.72 \end{bmatrix}, \vec{v_d} = \begin{bmatrix} 0.00 \\ 1.00 \\ 0.00 \\ 0.00 \end{bmatrix} \tag{3.2}$$

To be able to train the network it is necessary to measure how well it performed on the given training example. In other words, it is necessary to obtain some metric, a single value, that indicates if the network is classifying well, based on $\vec{v}$ and $\vec{v_d}$ alone. This value is called the cost, or loss, and different functions can be used to compute it, where the choice on which one to use depends on the desired application. One of the most commonly used functions in Artificial Neural Networks is the Mean Squared Error (MSE), that for a single training example is given by:

$$cost_0 = \sum_i (\vec{v_i} - \vec{v_{di}})^2 \tag{3.3}$$

which for the example in equation 3.2 would be $(0.89-0.00)^2+(0.21-1.00)^2+(0.07-0.00)^2+(0.72-0.00)^2 = 1.9395$. The cost in equation 3.3 is smaller when the network classifies the image correctly, and larger when the network doesn't know how to classify it properly.

No matter what function is chosen, for it to be used in neural network evaluation it must satisfy two properties: it must not be dependent on any other neuron's activation value besides the output layer neurons, and it must be able to be written as an average over all individual training examples, as in:

$$cost = \frac{1}{n_{images}} \sum_{k=0}^{n_{images}-1} (cost_k) \tag{3.4}$$

And it is this average over all images in the training data that will be the measure for the network performance on said data, in other words, the cost.

Because the only parameters that influence the results of the classification are the weights and biases of the network, these are the parameters that need to be tuned in order to improve the network performance, i.e. reduce the cost. And so the Cost Function will be a function that has as inputs all the weights and biases of the network and outputs the cost, based on all the images on the training data. The Cost Function could then be represented by:

$$C(w_1, w_2, ..., w_n) \tag{3.5}$$

where n is the total number of weights and biases. The next step is to discover what combination of input parameters brings the output of this function to a minimum.

3.2.2 Gradient Descent

Now that a Cost Function is explained it is time to show how to find its minimum. For any Deep Learning model the Cost Function will have a lot of inputs. For the example network of Figure 3.6 with 65025 neurons in the input layer, 5 neurons in each of the two hidden layers and 4 neurons in the output layer there are 325170 weights and 14 biases, and so its Cost Function will have 325184 inputs! And this is just a very small example network for concept explanation, real DNNs usually have millions of weights and biases. So for a better understanding of the methods to find the function's minimum it is better to start with examples of functions with fewer inputs.

To begin the explanation a Cost Function with only one input, like the one seen in Figure 3.8, is presented. The goal is to find the input value that minimizes the function. A good way to do this is to start with a random input value and check in which direction to step in order to lower the output. Refer to Figure 3.8 to better follow the explanation. If the slope (red line in Figure 3.8) of the function at the point is negative , then the input must step to the right (green arrows in Figure 3.8), and if the slope is positive then the input must step to the left (orange arrow in panel 3 of Figure 3.8). It is also possible to take these steps proportionally to the slope to avoid overshooting (length of arrows in Figure 3.8). By repeating this process multiple times the slope will eventually become flat,

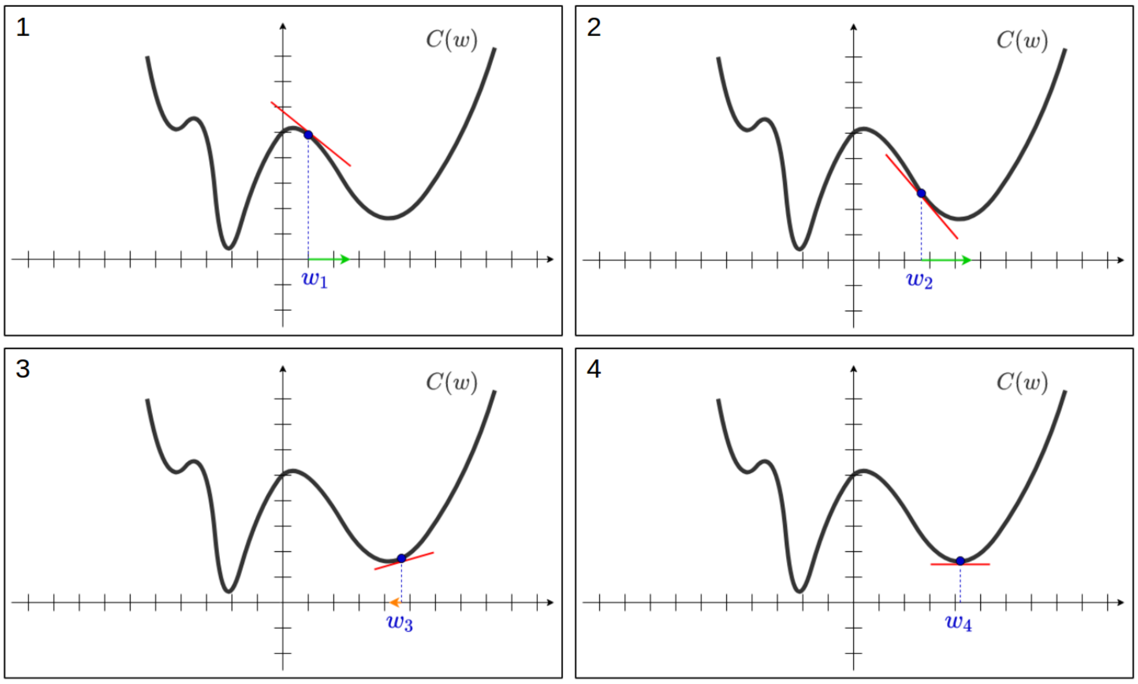


Figure 3.8: Finding a minima of a one input function. Panels 1 to 4 show the sequence of the method. First its measured the slope at a random initial input value (w_1) . If the slope is negative the input is shifted right (green arrow), and if the slope is positive the input is shifted left (orange arrow). The step size can be taken proportionally to the slope's absolute value (length of arrows). This step is repeated until a local minima is found (flat slope).

meaning that it reached a minimum value. This process can be imagined as releasing a ball at a random point of the function and it will eventually roll down, whichever direction that is. It is important to note that with this method the result will be a local minima, and not necessarily the global minima of the function, and this will also be true for the real DNN Cost Function. However this is not an issue in Deep Learning, as usually finding a local minima is more than enough to provide satisfying results.

If the complexity is increased by using a Cost Function with two inputs and one output, a similar line of thought can be used. In this case the input space can be seen as the xy plane, and the Cost Function will be a surface above it. In this example the starting point will also be random, but instead of computing a slope, what can be done is to determine in which direction in the input space can the point step in order to decrease the value of the function the fastest. The Gradient of the function can be used for this purpose. The Gradient of a function is a vector that indicates the direction to move in the input space, from the specified point, that provides the biggest possible increment on the output. In other words it points in which direction is the steepest increase. And so the negative of the Gradient gives the direction for the steepest decrease. The length of the vector also

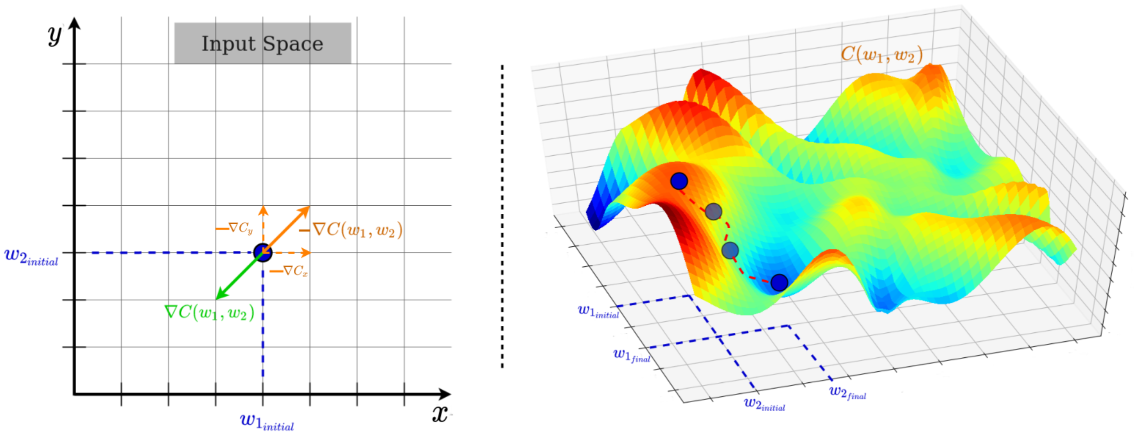


Figure 3.9: Finding a minima of a two input function. Left: The input space can be seen as the xy plane. Right: The Cost Function and path from an initial point to a local minima. The gradient of the Cost Function $\nabla C(w_1, w_2)$ at any input point gives the direction of the steepest ascent, and its negative indicates the direction of the steepest descent. By taking a small step in this direction and repeating this process for the next values, a local minima is achieved.

indicates how "steep" the steepest slope is. With this the algorithm to discover a minima of the Cost Function can be as simple as:

1. Compute the gradient $\nabla C(w_1, w_2)$;

2. Take a small step in the direction of $-\nabla C(w_1, w_2)$;

3. Repeat step 1 and 2 for the updated weights and biases until a minima is found;

This process of repeatedly adjusting the inputs of a function by a multiple of the negative of the gradient is called Gradient Descent. With the help of Figure 3.9 it is easier to visualize and understand the explanation of this method given in the previous paragraph.

It is important to note that the gradient vector has a component for each input, in other words, it is the combination of a vector in the x direction and a vector in the y direction, as seen on the left in Figure 3.9. Because of this the negative Gradient vector can be seen as a list of how *each* input of the function must be adjusted. And this is also true for the 325184 inputs Cost Function. If the values of all the weights and biases of the network are placed in a vector $\vec{W}$, then the vector $\nabla C(\vec{W})$ with the same number of elements, each being an indication on how the corresponding weight or bias in $\vec{W}$ must be adjusted, can be seen as:

$$\vec{W} = \begin{bmatrix} w_1 \\ b_1 \\ \vdots \\ w_n \\ b_m \end{bmatrix}, \nabla C(\vec{W}) = \begin{bmatrix} \frac{\partial C}{\partial w_1} \\ \frac{\partial C}{\partial b_1} \\ \vdots \\ \frac{\partial C}{\partial w_n} \\ \frac{\partial C}{\partial b_m} \end{bmatrix} \tag{3.6}$$

where the sign of each value in $\nabla C(\vec{W})$ indicates if the corresponding input in $\vec{W}$ must go up or down, and the magnitudes indicate which input changes will affect the cost the most. Taking a step in the direction of the negative of the function's gradient simply means updating the weights and biases in the following manner:

$$\vec{W}_{new} = \vec{W} - \eta \nabla C(\vec{W}) \tag{3.7}$$

where η is a constant usually referred to as the learning rate, which will determine the magnitude of the steps taken in the Gradient Descent (the lengths of the vectors in Figures 3.8 and 3.9).

What is important to retain is that by adjusting the weights and biases of the network with the Gradient Descent algorithm to reduce the Cost Function to a minimum, what it means is that the vector $\vec{v}$ in equation 3.2 will start to get closer to the vector $\vec{v}_d$ on the same equation, thus the network will be classifying the training data better. And because the cost involves an average over all the data in the training dataset (as seen in Equation 3.4), by minimizing it, it means that the network improves the performance on all of that data.

Because there are usually thousands and sometimes millions of images in the training dataset, the Gradient Descent algorithm becomes computationally expensive, since it needs to add up the influence of every single image at every single Gradient Descent step, so instead what is normally used is a variation of it called Mini Batch Gradient Descent.

In Mini Batch Gradient Descent the training data is divided into a whole number of mini batches, each containing only a small number of training data, preferably representative of all classes to classify into, and the Gradient Descent step is computed according to each mini batch. Because this method uses a lot less images per step, each step will not be as accurate to decrease the cost function, and some fluctuation on the error will be verified, but a minimum is found much faster than when using a real Gradient Descent step. When each mini batch is composed only of one image, the method is called Stochastic Gradient Descent.

Now that these concepts are explained it is now time to introduce the algorithm responsible to compute the Gradient Vector.

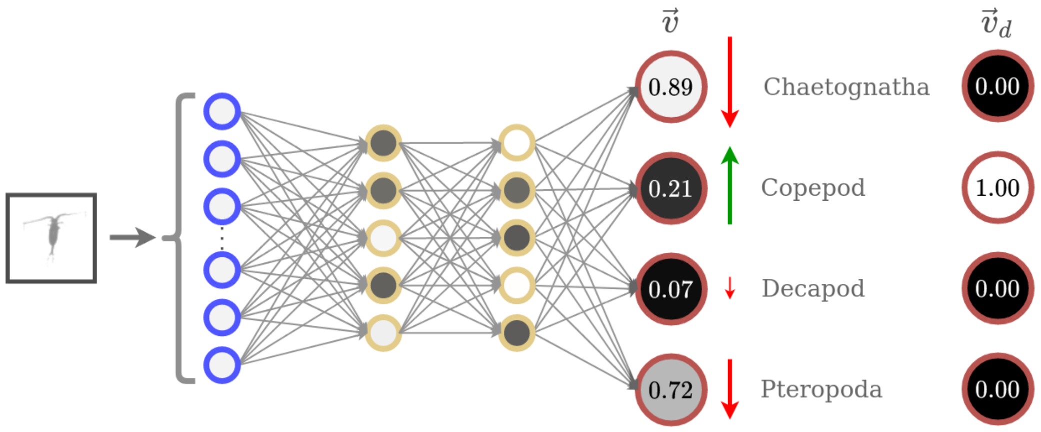


Figure 3.10: Given an input image of a copepod, an untrained network's output may differ largely from the desired output. The activations cannot be changed directly, but keeping track of in what way they should change is helpful. An increase of the copepod related neuron is more important than a decrease of the decapod related neuron, since the latter is already close to what it should be, hence why the arrow for the copepod neuron is larger than the arrow for the decapod neuron.

3.2.3 Backpropagation

The algorithm responsible for computing the gradient of the cost function is called Backpropagation. To begin the explanation on how it works the example network from Figure 3.6 will be used once again. In the case when the network is not trained at all, when it is fed with a single image of a copepod, the activations on the output layer can be similar to vector $\vec{v}$ in equation 3.2. These activations must be adjusted but can't be changed directly, the only way to change them is to adjust the weights and biases. However it is helpful to keep track of which adjustments should take place on the output layer, represented by arrows in Figure 3.10, relative to the desired output vector. Because in the example the desired classification is "Copepod", the value in the corresponding neuron should increase, while the values on all the other neurons should decrease, and the amount by how much they should change is visually represented by the arrow size. For instance, an increase on the neuron relative to the label "Copepod" is more important than a decrease on the neuron relative to "Decapod", because this last value is already close to what it should be, therefore the "Copepod" arrow is bigger.

Focusing the analysis only on the neuron which activation value should increase, the "Copepod" related neuron, its activation is given by:

$$a_{copepod}^{(L)} = g\left(\sum_i w_i^{(L)} a_i^{(L-1)} + b_{copepod}^{(L)}\right) \tag{3.8}$$

where L is only an indication of the layer the neuron belongs to and **not** an exponent, and g is the activation function. The objective is to understand how should the weights $w_i^{(L)}$, the bias $b_{copepod}^{(L)}$, or the activations of the previous layer $a_i^{(L-1)}$ change in order to increase the activation of the "Copepod" neuron. For explanation purposes the MSE function will be used for the cost measure, but others could be used.

Starting with the weights: from equation 3.3 the cost for a **single** image $C_0(\vec{W})$, focusing only on the "Copepod" neuron related component, is given by:

$$C_0(\vec{W}) = (a_{copepod}^{(L)} - y)^2 + ... \tag{3.9}$$

where y is the desired output for this neuron (in this case y would be 1 for the copepod image). Then, for ease of comprehension, if the weighted sum and bias are given a new name $z_{copepod}^{(L)}$, such as:

$$z_{copepod}^{(L)} = \sum_i w_i^{(L)} a_i^{(L-1)} + b_{copepod}^{(L)} \tag{3.10}$$

then the activation is given by:

$$a_{copepod}^{(L)} = g(z_{copepod}^{(L)}) \tag{3.11}$$

The objective now is to understand how a small change in a single weight connected to the neuron, i.e. $\partial w_i^{(L)}$, affects the cost C_0, or rather, to compute the ratio:

$$\frac{\partial C_0}{\partial w_i^{(L)}} \tag{3.12}$$

Because a change in $w_i^{(L)}$ will produce a change in $z_{copepod}^{(L)}$ (equation 3.10), which itself will cause a change in $a_{copepod}^{(L)}$ (equation 3.11), which finally will cause a change in C_0 (equation 3.9), the following chain rule can be written:

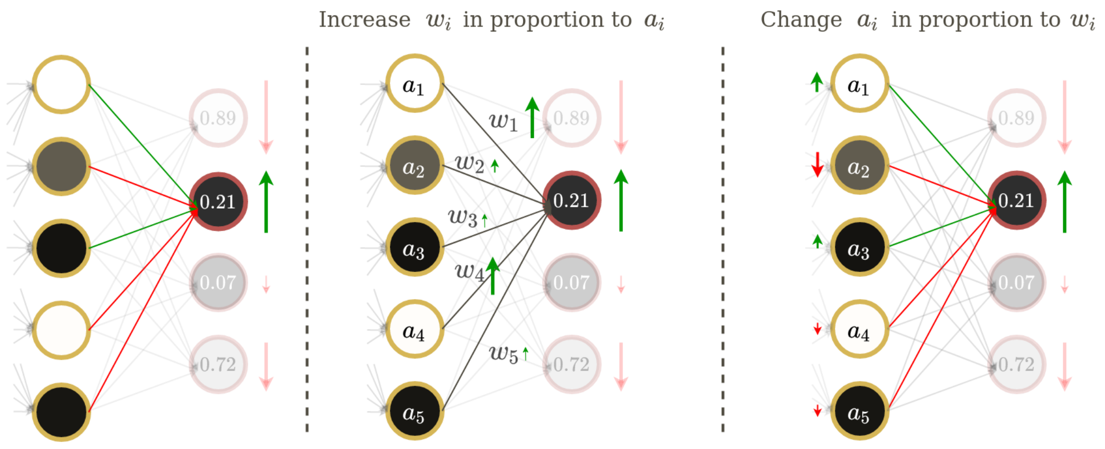
$$\frac{\partial C_0}{\partial w_i^{(L)}} = \frac{\partial z_{copepod}^{(L)}}{\partial w_i^{(L)}} \cdot \frac{\partial a_{copepod}^{(L)}}{\partial z_{copepod}^{(L)}} \cdot \frac{\partial C_0}{\partial a_{copepod}^{(L)}} \tag{3.13}$$

By computing the relevant derivatives:

$$\frac{\partial C_0}{\partial a_{copepod}^{(L)}} = 2(a_{copepod}^{(L)} - y) \tag{3.14}$$

$$\frac{\partial a_{copepod}^{(L)}}{\partial z_{copepod}^{(L)}} = g'(z_{copepod}^{(L)}) \tag{3.15}$$

$$\frac{\partial z_{copepod}^{(L)}}{\partial w_i^{(L)}} = a_i^{(L-1)} \tag{3.16}$$



Increase w_i in proportion to a_i
Change a_i in proportion to w_i
0.89
0.21
0.07
0.72
a_1
a_2
a_3
a_4
a_5
w_1
w_2
w_3
w_4
w_5

Figure 3.11: The output neuron is connected to the previous layer by positive (green lines) and negative (red lines) weights (left). To increase its activation one can: increase the weights connected to them proportionally to the activations on the previous layer (center, weights connected to brighter neurons should increase more); change the activations in the previous layer, activations of neurons connected to negative weights should decrease, and the ones connected to positive weights should increase, and these changes should be proportional to the absolute value of the weights connecting them (right).

and substituting in equation 3.13, the equation becomes:

$$\frac{\partial C_0}{\partial w_i^{(L)}} = a_i^{(L-1)} g'(z_{copepod}^{(L)}) 2(a_{copepod}^{(L)} - y) \tag{3.17}$$

It gets obvious from equations 3.14 and 3.17 that the change in the cost is proportional to the difference between the current activation value and the desired output. When the neuron's activation is farther from what it should be, even small increases in a weight connected to it would have a high impact on the cost function. This is where the sizes of the arrows in Figure 3.10 comes from. It is also obvious from equation 3.16 that the amount that a small change in a weight influences the neuron's activation depends on the activation value of the neuron it is connected to from the previous layer, and so the increase in weights connected to higher activations neurons would be greater than the ones connected to nearly inactive neurons. This is reminiscent of a theory in neuroscience for how biological neurons learn, Hebbian theory [38], that is often phrased as "Neurons that fire together wire together", and is visually represented in the center panel of Figure 3.11.

The method to determine how the "Copepod" neuron's bias should change is similar to the weights, the only way it differs is replacing $\partial w_i^{(L)}$ with $\partial b_{copepod}^{(L)}$ in the chain rule in equation 3.13:

$$\frac{\partial C_0}{\partial b^{(L)}_{copepod}} = \frac{\partial z^{(L)}_{copepod}}{\partial b^{(L)}_{copepod}} \cdot \frac{\partial a^{(L)}_{copepod}}{\partial z^{(L)}_{copepod}} \cdot \frac{\partial C_0}{\partial a^{(L)}_{copepod}} \tag{3.18}$$

and since the partial derivative of $z^{(L)}_{copepod}$ in respect to $b^{(L)}_{copepod}$ is 1, the chain rule equation becomes:

$$\frac{\partial C_0}{\partial b^{(L)}_{copepod}} = g'(z^{(L)}_{copepod})2(a^{(L)}_{copepod} - y) \tag{3.19}$$

The third way the "Copepod" neuron could increase is by changing the activations of the neurons in the previous layers, and even though they can't be changed directly it is important to keep track on how they should change, the same way it was important to know how the output neurons should change in Figure 3.10. Like so, the way a change in the activation on a given neuron from the previous layer, i.e. $a^{(L-1)}_i$ affects the cost, will be represented by the chain rule:

$$\frac{\partial C_0}{\partial a^{(L-1)}_i} = \frac{\partial z^{(L)}_{copepod}}{\partial a^{(L-1)}_i} \cdot \frac{\partial a^{(L)}_{copepod}}{\partial z^{(L)}_{copepod}} \cdot \frac{\partial C_0}{\partial a^{(L)}_{copepod}} \tag{3.20}$$

and because:

$$\frac{\partial z^{(L)}_{copepod}}{\partial a^{(L-1)}_i} = w^{(L)}_i \tag{3.21}$$

the equation becomes:

$$\frac{\partial C_0}{\partial a^{(L-1)}_i} = w^{(L)}_i g'(z^{(L)}_{copepod})2(a^{(L)}_{copepod} - y) \tag{3.22}$$

From equations 3.21 and 3.22 it is obvious that how the cost is affected with small changes in the neuron's $a^{(L-1)}_i$ activation is proportional to the weight connecting it to the "Copepod" output neuron. This means that to decrease the Cost Function the fastest, the activations in the previous layer should change proportionally to the value of the weight connecting them to the output neuron. Furthermore, neurons in the previous layer connected by weights with negative values should decrease their activation and neurons connected by weights with positive values should increase them. This is visually represented in the right panel of Figure 3.11.

But note that not only the "Copepod" neuron has an "opinion" on how the neurons in the previous layer should change, as all output neuron's activations affect the cost, as seen in equation 3.3. The neurons which activations should decrease also have a say in this because they are also influenced by the activation values of the previous layer, and so all their "opinions" are summed, leaving the chain rule for a neuron in the previous layer to be:

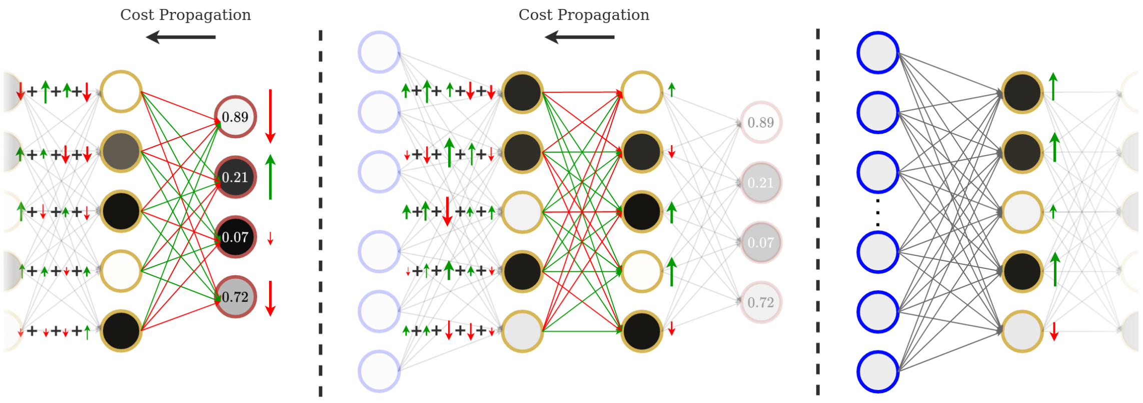


Figure 3.12: Representation of Backpropagation. Left: each neuron in the output layer has an "opinion" on how the neurons in the previous layer should change, based on equation 3.20. Center: these "opinions" are added together and can now be used to compute the Gradient Vector's components related to this layer's weights and biases, and the process can be repeated for previous layers. Right: When the last hidden layer is reached and the corresponding gradients are computed, there is now an idea on how all weights and biases in the network should change. This propagation of the cost backwards, from last to first layer is what gives the algorithm its name.

$$\frac{\partial C_0}{\partial a_i^{(L-1)}} = \sum_{j=0}^{n_L-1} \left(w_{ji}^{(L)} g'(z_j^{(L)}) 2(a_j^{(L)} - y_j) \right) \tag{3.23}$$

with n_L being the number of neurons in layer L (in this case the output layer) and w_{ji} are the weights connecting the neuron i in the previous layer to the neurons j in the output layer.

Once it's known how sensitive the cost function is to the activations in the previous layer, there is now a metric for how each neuron on the previous layer should change, the same way there was with the output layer in Figure 3.10, and so the corresponding weights and biases can now be computed and updated, and the process can be repeated for the layers before this one, propagating the cost backwards, as seen in Figure 3.12. And this is where the algorithm Backpropagation gets its name from.

And so, generalizing for any network with any activation function $g(z)$ and Cost Function $C(\vec{W})$, the way a weight connecting a neuron j in layer l to a neuron i in layer $l-1$, and the bias of neuron j should change is given by:

$$\frac{\partial C_0}{\partial w_{ji}^{(l)}} = a_i^{(l-1)} g'(z_j^{(l)}) \cdot \frac{\partial C_0}{\partial a_j^{(l)}} \tag{3.24}$$

and,

$$\frac{\partial C_0}{\partial b_j^{(l)}} = g'(z_j^{(l)}) \cdot \frac{\partial C_0}{\partial a_j^{(l)}} \tag{3.25}$$

where for the neurons on the output layer $\frac{\partial C_0}{\partial a_j^{(l)}}$ is the partial derivative of the chosen Cost Function with respect to $a_j^{(l)}$, which for MSE is given by:

$$\frac{\partial C_0}{\partial a_j^{(l)}} = 2(a_j^l - y_j) \tag{3.26}$$

and:

$$\frac{\partial C_0}{\partial a_j^{(l)}} = \sum_{k=0}^{n_{l+1}-1} \left(w_{kj}^{(l+1)} g'(z_k^{(l+1)}) \cdot \frac{\partial C_0}{\partial a_k^{(l+1)}} \right) \tag{3.27}$$

for the neurons in the hidden layers.

Of course, the explanation so far was referring to a single image only, but because of how the real cost was defined in equation 3.4, for the real Gradient Vector's parameters computation, it takes an average over all n training images in the dataset (or in the mini batch if using Mini Batch Gradient Descent), and so said parameters are given by:

$$\frac{\partial C}{\partial w_{ji}^{(l)}} = \frac{1}{n} \sum_{k=0}^{n-1} \frac{\partial C_k}{\partial w_{ji}^{(l)}} \tag{3.28}$$

$$\frac{\partial C}{\partial b_j^{(l)}} = \frac{1}{n} \sum_{k=0}^{n-1} \frac{\partial C_k}{\partial b_j^{(l)}} \tag{3.29}$$

Note how these are the parameters of $\nabla C(\vec{W})$ as defined in equation 3.6, just with a different notation, thus concluding how the Backpropagation algorithm works, and all the steps and concepts necessary to understand how an ANN is trained are now explained.

3.2.4 Training an Artificial Neural Network

The process of training an ANN becomes fairly straight forward once the concepts explained in the previous sub-chapters are understood. It simply consists of four different moments: the forward pass, the cost computation, the backward pass and the weight update. During the forward pass the training data is passed through the network in batches for it to predict the classification results. Then the cost computation takes place based on the classification results and the desired results as defined by the Cost Function. With the obtained cost it is then possible to compute the Gradient Vector ∇C in the backward pass, using the Backpropagation algorithm. Finally the network parameters, the weights and the biases, can be updated by a multiple of the Gradient Vector, as seen in the Gradient Descent method, in Equation 3.7. By repeating these four steps a given amount of iterations, the network should start to improve the classification results on the training dataset.

3.2.4.1 Datasets

If the training runs for too long it is possible that the network starts to pick up specific features from the training data that are not necessarily inherent to the classes on the dataset but rather residual variation, or noise, that the model can't distinguish from actual meaningful features, and although its accuracy improves on the training dataset, it fails to generalize for new data that the network has never seen before, thus leading to low accuracy on new input data. This is a problem, because the objective of training is to be able to generalize to new data. This problem when the model fits its parameters too specifically for the training data, disregarding actually relevant class features, achieving high accuracy on training data but low accuracy on new data, is called overfitting, and can occur for multiple reasons, from insufficient training data, to too complex models, but also from overtraining. To avoid overtraining the network, and to obtain an unbiased network evaluation metric, what is usually done is split all labeled data in two different datasets: the training dataset and the validation dataset. The training dataset will be used for actual training, to update the network parameters, and the validation dataset will be used for an unbiased accuracy evaluation only, and won't have any effect on the weights and biases updates. This second dataset is important both to see how the network generalizes to new data and to serve as indication for when to stop training, as training should stop when the accuracy on the validation dataset stops increasing, even if the accuracy on the training dataset is still improving. A third dataset called test dataset is used when evaluating between different network architectures, or different training configurations. How the data is split between these datasets is dependent on the amount of data available and on the type of model being trained, but a common rule of thumb, or a good starting point, is to split it as 80% for the training dataset and 20% for the validation dataset when evaluating only a single model and training process, or 10% validation dataset and 10% test dataset, when evaluating multiple models, or a single model with different training processes.

For the training to be effective it is also necessary to define a set of parameters that unlike the weights and the biases are not automatically learned by the network during training, such as the number of iterations, the batch size or the learning rate. These parameters that are not automatically learned during training are commonly referred to as hyperparameters, and are set before training starts.

3.2.4.2 Hyperparameters

Hyperparameters are both the variables that define the network structure and the ones that define how the network is trained.

The hyperparameters related to network structure are the type and number of layers, the number of neurons, the activation functions, and the weights and biases initialization method, to name a few. The type and number of layers and neurons is important for obvious reasons as it was already mentioned earlier in this chapter. Shallow networks with few layers and neurons will easily lead to underfitting the data, that is, they won't be able to find the relationship between the input data and the desired output, therefore leading to low accuracy. By increasing the depth of the network it is possible to increase accuracy to an extent, with the trade off being that it becomes more computationally expensive to train.

The choice of the activation functions is also important as they are responsible to introduce non-linearity to the models, allowing it to learn non-linear prediction boundaries on the data. As explained earlier in this chapter, the choice of the activation function can be dependent on the task, so for instance the ReLU could be used in the hidden layers of a network, being the most popular choice, and a sigmoid function could be used in the output layer if the task is binary classification, as it outputs a value between 0 and 1 (that can be seen as the probability for the input data to belong to the chosen class).

Weight initialization is also of utmost importance. If the weights and biases are initialized to 0 then every neuron's activation will also be 0 at each iteration independently of the input, and therefore they will all follow the same gradient, thus leading to all neurons learning the same features. This is also a problem if the weights are all initialized with equal values, as with the weights and biases being all the same, so will the influence of each neuron in the cost be the same, leading to identical gradients, and so all the weights would be updated by the same amounts. This is known as the network failing to break symmetry. To avoid this it is necessary to initialize the weights with random unequal values. This will break symmetry, but other problems may arise. Initializing the weights with too large values will lead to exploding gradients, that is, the gradients start to get larger and larger as backpropagation advances from the output layer to the input layer, causing very large weight updates resulting in cost oscillation around the minimum value, or even gradient descent divergence. On the other hand, initializing the weights with too small values will lead to the vanishing gradients problem, that is, the gradients start to get smaller and approaching zero as backpropagation advances, leaving the weights of the initial layers nearly unchanged, leading to convergence of the cost before it reached a minimum value. To summarize, the weights and biases must be initialized with unequal values to break symmetry, with small enough values so there aren't exploding gradients, but not too small to avoid the vanishing gradient problem. There are several methods for weight initialization, with uniform distribution of random values being one of the most commonly used.

The hyperparameters related to training are the ones that define the behaviour of the training moment itself, for how long it should run, how the cost optimizer should behave while training, and so on. Examples of these are the batch size, the number of epochs and the learning rate. The batch size was already mentioned earlier, and is the number of training samples present in each mini-batch to be fed at each iteration during gradient descent. Larger batch sizes lead to more accurate update steps, but training becomes both computationally expensive, as more computer memory is needed at each step to store the effects of every item in the batch, and slow. With smaller batch sizes it is still possible to reach a minimum, and faster, but each gradient descent step will not be as accurate.

An epoch is the term used for when the entire training dataset has been fed to the network during training exactly one time, and it is commonly used to set the duration of training. Although in pactice the duration of training is usually set by the number of iterations, the number of epochs is used both for reference and to calculate the equivalent number of iterations, as there is a relation between them. This relation also includes the batch size, because the number of iterations needed to complete an epoch is obviously dependent on the number of data samples per iteration. So for instance if the training dataset contains 10000 samples, and the batch size is 10, then it takes 1000 iterations to complete an epoch. From here it is possible to calculate how many iterations are needed for a given number of epochs with the great power of mathematics. As mentioned previously, the number of epochs must be high enough to achieve good validation accuracy, but not too high, to avoid overfitting.

The choice of a good learning rate is important because a too small value will result in a very slow training process, while a too large value may result in overshooting of the cost, or even gradient descent divergence, therefore making it impossible to reach a cost minimum. Sometimes some strategies that change the learning rate as training advances are adopted, where training starts with a larger learning rate in order to converge faster in the direction of the cost minimum, and then is gradually reduced as the cost starts to decrease.

There are even other optimization methods that update the weights differently from Gradient Descent, such as AdaDelta [39], AdaGrad [40] and Adam [41]. However, all these methods are also gradient based and the difference they introduce on the overall explanation on how Artificial Neural Networks work is on the equation used to update the network parameters, with each introducing its own set of hyperparameters. Therefore it's comprehensible that they are only mentioned for completeness of information on the topic, as the explanation of these methods goes far beyond the scope of both this thesis and this chapter, which is meant to provide a basic, although solid and extensive as may be, idea on the basics of Neural Networks. What is, however, important to explore further is a special type of Neural Network that sparked a new interest in Machine Learning in the last decade, given its remarkable achievements in computer vision applications, dominating at the time of this writing the state-of-the-art on image classification, becoming one of the most important achievements in Deep Learning, the Convolutional Neural Networks.

3.3 Convolutional Neural Networks

Regular Neural Networks like the ones presented previously can achieve good results in many classification tasks, but when the inputs for classification are images they become too computationally expensive to use. For a 255x255 image for instance, a single neuron in the first hidden layer would have 65025 weights connecting it to the input neurons. As images are also commonly represented by three different channels, a red, a green and a blue, this would mean that the number of weights for a single neuron would actually be three times this value. Considering that for the network to be able to get good accuracy it must be complex enough to avoid underfitting, that is, it must have a good amount of neurons per hidden layer, as well as a good amount of hidden layers, it is easy to comprehend that the number of weights to train becomes too large to compute. Convolutional Neural Networks however are designed with the assumption that the inputs are images, allowing to encode a few properties into their design that make them more efficient, by largely reducing the amount of learnable parameters in the network. They work with volumes of neurons, with their layers arranged in three dimensions, a width, a height, and a depth. Each layer in a CNN receives as inputs 3D volumes of neurons and transform them into a new 3D volume output, as seen in Figure 3.13. The input layer will have the same width and height as the input image, and its depth will be the number of channels of the image (for an RGB image it is 3, for a gray scale image it is 1), with each neuron's activation value being the corresponding pixel value at the exact same location. The output layer will contain an output value per class, as seen earlier.

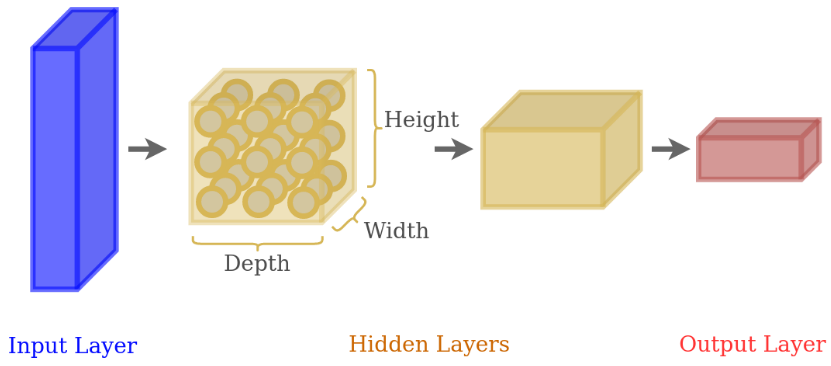


Figure 3.13: Representation of a Convolutional Neural Network, where each layer is a 3D volume of neurons.

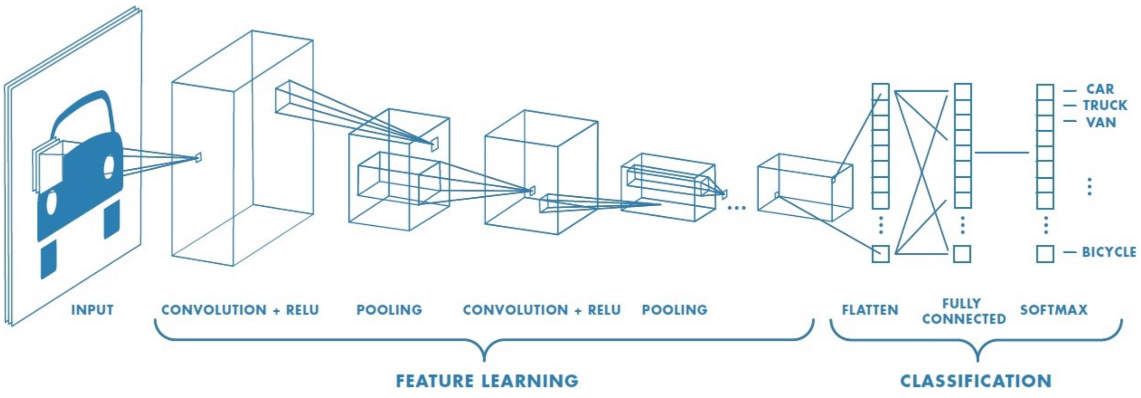


Figure 3.14: Convolutional Neural Networks are divided in two parts, the feature learning part and the classification part (From [42]).

Convolutional Neural Networks are divided in two parts, one responsible for feature learning and another responsible for the actual classification. This is achieved with a set of special layers for each task, as seen in Figure 3.14. But before getting into detail on how these layers work it is necessary to understand the concept that makes these networks great for image classification, an operation that gives them their name, the convolution.

3.3.1 Convolutions

Convolutions in image processing is an operation that transforms the input image into another different image by means of a mathematical convolution of image data and a kernel. The way it works is by sliding the kernel matrix over the input image, that can be seen as a matrix of pixel values, multiplying overlapping values of the two matrices and summing it up to generate the corresponding output value. As the kernel slides through the image it generates the output matrix. This process is better visualized in Figure 3.15, where the kernel, a 3x3 matrix with ones in the diagonals and zeros elsewhere, slides through the 5x5 input to generate the output matrix.

It is visible in Figure 3.15 that after the input got convolved with the kernel, the size of the output got reduced. It is possible to keep the dimensions of the input by bordering it

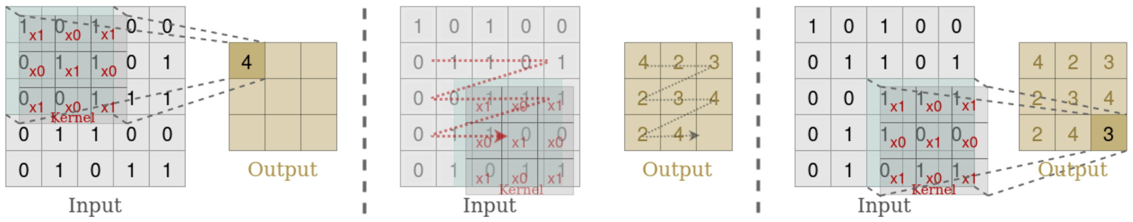


Figure 3.15: Convolution of an input matrix with a 3x3 kernel. As the kernel slides through the input, it generates the output matrix.

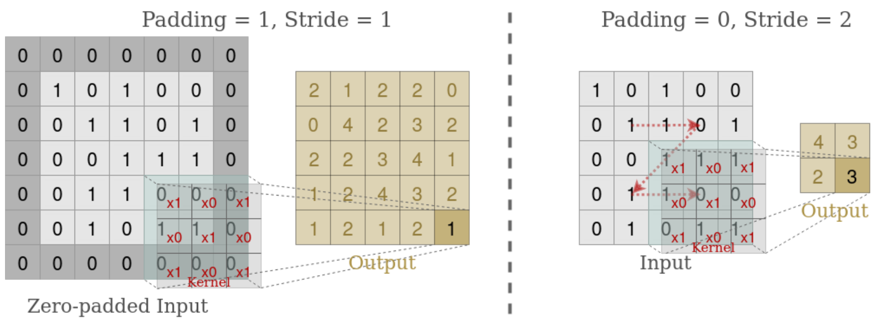


Figure 3.16: Example of padding and stride. Left: zero-padding is used to maintain the same dimensions of the input in the output, with a stride of 1. Right: no padding, a stride of 2 is used to further reduce dimensions in the output.

with zeros all around and then applying the kernel to this augmented input. This process is called padding, or zero-padding to be more precise, and the number of zeros to pad the image with is dependent on the kernel size: for a 3x3 kernel it is only necessary to pad the input once, while with a 5x5 kernel it is necessary to pad it twice, etc. It is also possible to deliberately reduce the output size in relation to the input by making the kernel move multiple pixels to the right and down while sliding. The number of pixels to shift over the input is called stride, and a stride of 1 will make the kernel shift a pixel at a time, a stride of 2 will make the kernel shift 2 pixels at a time, and so on. In Figure 3.16 examples of padding and stride are shown.

With different kernels it is possible to transform the image in multiple ways, from blurring, to sharpening, to edge and corner detection, as seen in Figure 3.17. This is what makes this operation so important to CNNs, because applying multiple different kernels to an image can highlight multiple different features of the object in the image that can be helpful to distinguish it from objects of different classes, e.g. if the kernels evidence two round wheel-like objects it may be a bicycle, but it probably isn't a fridge.

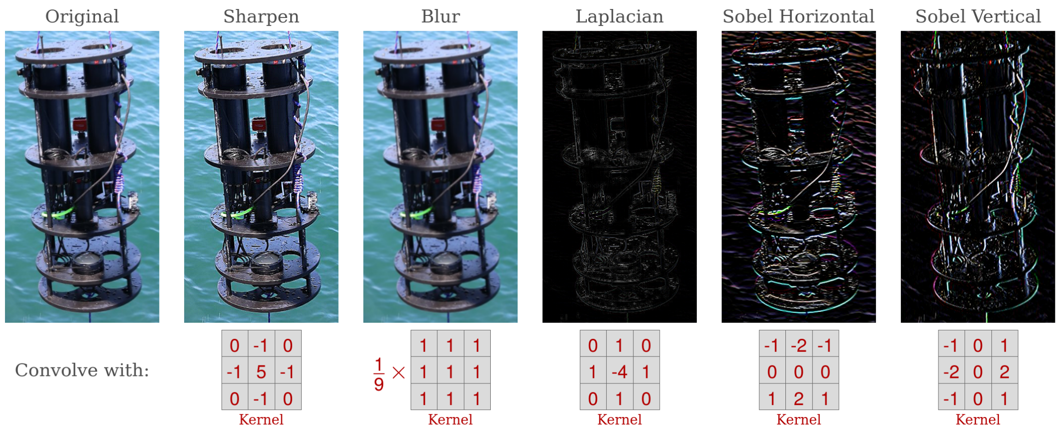


Figure 3.17: Convolution of the same image with different kernels.

3.3.2 Layers in a CNN

There are three main types of hidden layers that are used to build a Convolutional Neural Network: Convolutional Layers, Pooling Layers and Fully Connected Layers. The first two are responsible for the feature extraction part of the network, while the third is responsible for the classification itself.

3.3.2.1 Convolutional Layers

Convolutional Layers are the core building block of CNN and their objective is to detect features from the input. To achieve this, Convolutional Layers have two properties that are behind the success of CNNs in both image classification and parameter reduction: local connectivity and parameter sharing.

Local connectivity is related to the way the neurons in a Convolutional Layer are connected to the neurons in the previous layer. Unlike regular Neural Networks, neurons in this type of layers are not connected to all neurons in the layer before it. Instead, each neuron in Convolutional layers is only connected to neurons inside a small volume with dimensions FxFxD_1, with F being a hyperparameter usually referred to as receptive field or filter size, and D_1 is the depth of the layer it is connected to. Neurons on the same depth slice in a convolutional layer (i.e. neurons in the same depth coordinate along width and height of the layer) are all connected to volumes of neurons of the same dimension, shifted along the input's width and height by a hyperparameter stride S, similarly to how it happens in image convolutions explained earlier. The depth of a Convolutional Layer is given by a hyperparameter K, referred to as the number of filters, and all neurons at the same width and height location along the depth, also known as depth column or fibre, are connected to the same neurons in the previous layer. All these concepts are better visualized in Figure 3.18.

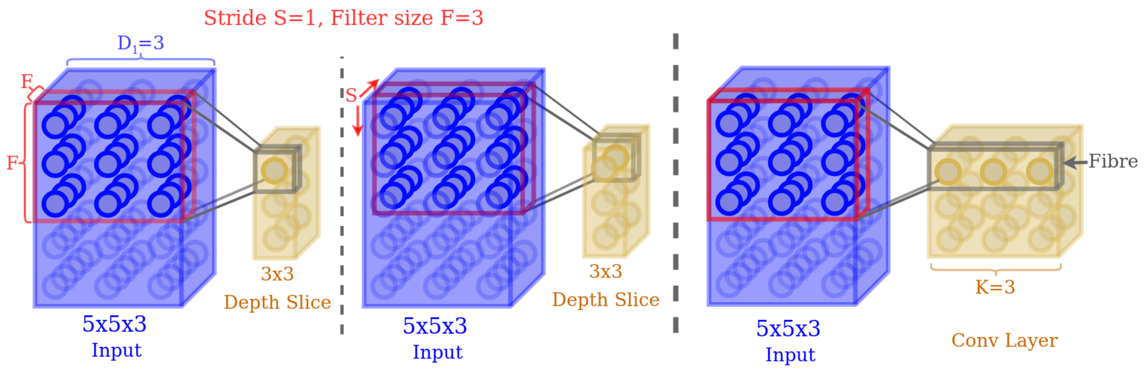


Figure 3.18: Neurons in the same depth slice are connected to FxFxD_1 regions, with a stride S along the input's width and height. The Conv Layer's depth is given by hyperparameter K. Neurons in a fibre are connected to the same inputs.

With Local Connectivity a neuron in the Convolutional Layer introduces FxFxD_1 weights and a bias parameter. Considering for example an input image of dimension 255x255x3, a neuron in a Convolutional Layer connected to it with a filter size of 3 will generate 27 weights and a bias, opposed to the 195075 weights if it was connected in a fully connected manner.

Of course local connectivity alone won't reduce the number of parameters in the network enough for it to become computationally feasible in real architectures. Each neuron may introduce less connections than when fully connected to the input, but if the layer it is on is composed of hundreds of thousands of neurons, which is not uncommon, it will still introduce too many parameters. Consider the first Convolutional Layer in Jialun Dai's ZooplanktoNet [28] mentioned in Chapter 2, with dimensions 55x55x96, which locally connects to the 227x227x1 input layer with a receptive field of 11. This layer has 290400 neurons, each connected to 11x11 neurons in the input, introducing 121 weights and a bias per neuron in the Conv Layer, resulting in a total of 35428800 parameters introduced by the layer, which is still too much.

Parameter sharing is a property applied in Convolutional Layers to vastly reduce the amount of parameters in the network. With parameter sharing all neurons in the same depth slice share the same weights and bias. It may seem counter-intuitive at a first glance to have whole sets of neurons sharing the same parameters in the layer, but it will all make sense in a few lines. Back to the layer presented in the last paragraph, which consists of 96 55x55 depth slices, by applying the parameter sharing scheme to it, the number of unique parameters comes down to 96x11x11 weights and 96 biases, since all 55x55 neurons in each depth slice will use the same parameters, resulting in a grand total of 11712 parameters. Using parameter sharing dropped the amount of parameters to store in memory for this layer by 99.97%!

At this point, the most attentive reader may be realizing that there is something familiar in the way that Convolutional layers interact with their input layer. The clues are all there: the inputs to the network are images; neurons are connected to small FxF regions on the input along its full depth; on the same Conv layer's depth slice the neurons are connected to regions defined by a stride along the input width and height; neurons on the same depth slice all use the same weight and bias values; the activation of a neuron is obtained by multiplying the values of the neurons it is connected to by the corresponding weights and then added together, along with a bias value; They are called Convolutional layers... That's it! Convolutional Layers operate on the input images exactly the same way as image convolutions work! In practice it is actually performing image convolutions with their weights as seen in Figure 3.19, hence where they get their name from. This is also why the sets of weights in Conv layers are usually called filters or kernels, and why the output of the layer is also called a feature map, and why the hyperparameter K is the number of filters.

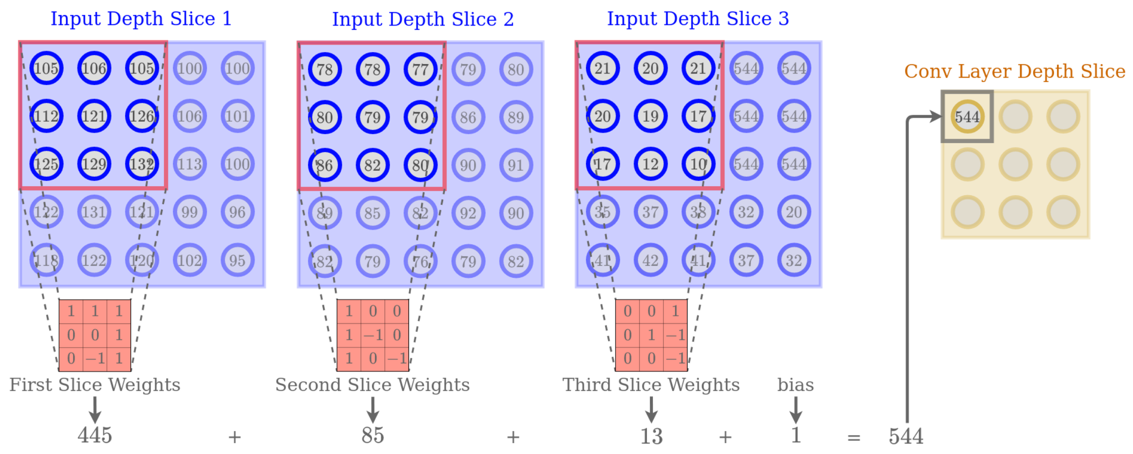


Figure 3.19: 2D visualization of left panel in Figure 3.18. In Convolutional layers the weights behave like 3D kernels in image convolutions, thanks to local connectivity and parameter sharing.

So when the network is training, each depth slice is actually learning a 3 dimensional filter to apply to the image, so one may detect horizontal lines, other can learn to detect vertical lines, blobs of colours or any other feature, as seen in Figure 3.20, and the output of each depth slice, or its activation map, is the convolution result, that as seen before are themselves images. This also explains why parameter sharing is reasonable. It is based on the assumption that if detecting any type of feature is important at a specific location in the image, it is also important to detect it at any other location. If a horse is at the center of an image, or on the left or right, far away or close, a network is still expected to correctly classify it as a horse if trained to do it. As in image convolutions, it is also possible to define a padding hyperparameter P in Conv Layers to maintain dimensionality (along width and height) of the input.

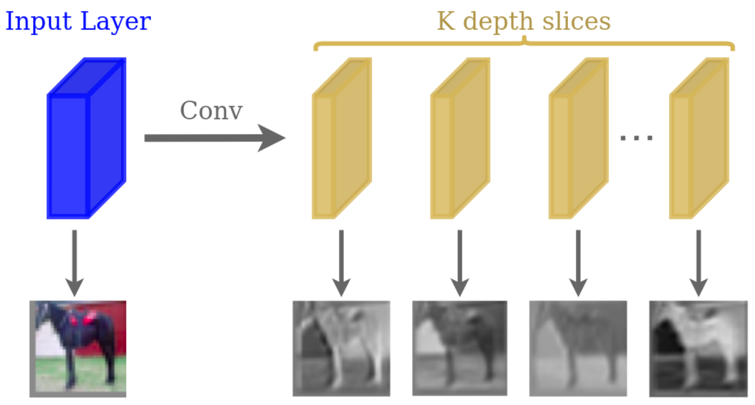


Figure 3.20: Each depth slice in a Convolutional layer is learning a 3 dimensional filter to convolve with the input, and its activation map is the convolution result.

As more and more Convolutional layers are added deeper in the network the complexity of the features detected increase, from simple shapes in shallow layers, like edges and colour blobs, to text, full patterns or actual face features in deeper layers. This is visible in Figure 3.21 from the work of Matthew D. Zeiler and Rob Fergus, ”Visualizing and Understanding Convolutional Networks” [43]. What is represented here is the top 9 images that provide the strongest activations for a random activation map at each layer, alongside the highlighted features that are responsible for such activations. For example in layer two on the right are two groups of the 9 images that activate the corresponding activation map the most, and on the left is a projection of the pixels that have the most impact on such activations. Note that the blocks on the left in layers 1, 2 and 3, and on the bottom of layers 4 and 5 are not the actual activation maps of some depth slice, but rather projections of the pixels from the input images (on their right in layers 1, 2 and 3, and above them in layers 4 and 5) that influence the most the activations for a given feature map, therefore, the types of features that the feature maps actually detect. Here it is possible to observe the hierarchy of features in the network: from simple lines and colour patches that are detected in the first layer, to complex and diverse types of features such as animal eyes, logos and areas with text, and grass in images. It is also visible that in shallow layers each group of 9 images is fairly consistent in content, whereas on deeper layers there is much more variance on the 9 images that activate the corresponding feature map, e.g. layer 5, the group on the right displays images that are apparently not related whatsoever class-wise, but what causes strong activations is the grass patches in the background in different regions of the image, and not the foreground ”class” object.

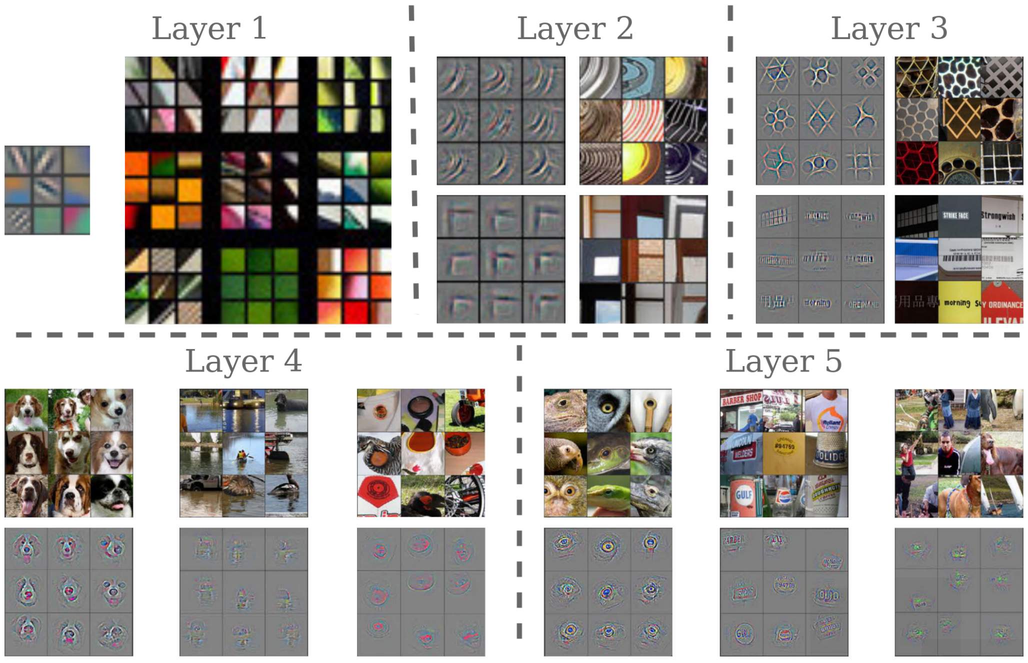


Figure 3.21: Visualization of detected features in a trained CNN, from [43].

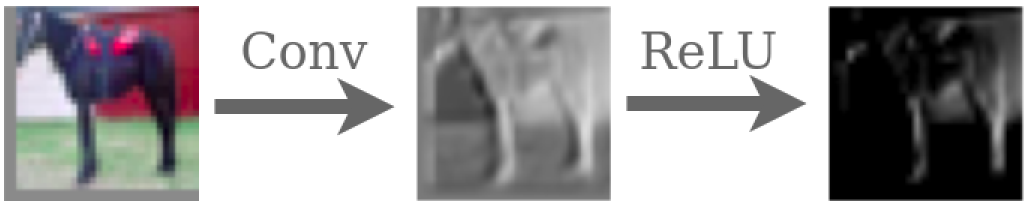


Figure 3.22: Effect of a ReLU on an activation map.

For Convolutional Neural Networks the most commonly used activation function is the ReLU, to introduce non-linearity. It is a good choice, since in the case of CNNs it makes sense to expect non negative outputs, given their image related nature, and this function converts all negative outputs to 0, while keeping the positive values unchanged. The effects of a ReLU function on an activation map is seen in Figure 3.22.

To conclude, Convolutional Layers:

- Require setting of four hyperparameters: the number of filters K, the filters size F, the stride S and the amount of zero-padding P;
- Receive as inputs volumes with dimensions $W_1 \times H_1 \times D_1$ and output a volume of size $W_2 \times H_2 \times D_2$, where $W_2 = (W_1 - F + 2P)/S + 1$, $H_2 = (H_1 - F + 2P)/S + 1$ and $D_2 = K$;
- By using parameter sharing, it introduces $(F \cdot F \cdot D_1) \cdot K$ weights and K biases;
- Each depth slice in the output is the result of a convolution of the input with its learned filter with a stride S, offset by a bias;

3.3.2.2 Pooling Layers

Pooling Layers are used to reduce the spatial size of the input volume, in order to reduce the number of trainable parameters and computation in the network, which also helps to control overfitting, as it reduces model complexity. The way it works is with FxF filters, typically 2x2, applied over each depth slice of the input volume independently, with a defined stride value S. These filters apply a function to the overlapped values in the input in order to output a value per overlapped area, effectively reducing the feature maps dimensions while keeping the most important information. The most common types of pooling are average pooling, which returns the average value of the overlapped area, and max pooling, which returns the maximum value in said area, with max pooling being the preferred choice. In Figure 3.23 it is possible to visualize this process using max pooling with a 2x2 filters with a stride of 2 over a 4x4 depth slice, and the result of such pooling to an input layer.

Since they apply a fixed function to their inputs, pooling layers don't introduce learnable parameters, their sole purpose is to reduce the network's parameters by discarding

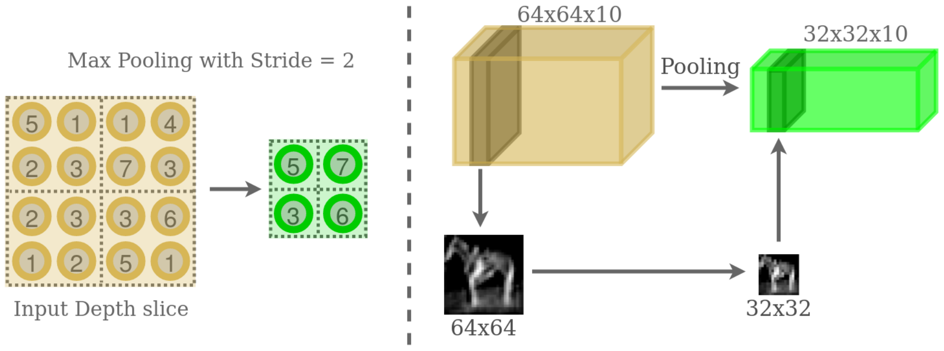


Figure 3.23: Pooling layer representation. Left: Max pooling operation with 2x2 filters applied with a stride of 2 over a depth slice. Right: results of such pooling over a 64x64x10 input volume resulting in a 32x32x10 pooling layer.

some activations. In fact a pooling layer with 2x2 filters applied with a stride of 2 will discard 75% of its input layer. Typical applications employ strides of 2 and filter sizes of 2, or 3 in some cases, but larger values are not used as they become too destructive. To summarize, pooling layers:

- Can have various types, like max pooling or average pooling, with max pooling being the preferred choice;
- Require setting of three hyperparameters: the type of pooling, the stride S and the filter size F;
- Receive as inputs volumes with dimensions $W_1 \times H_1 \times D_1$ and output a volume of size $W_2 \times H_2 \times D_2$, where $W_2 = (W_1 - F)/S + 1$, $H_2 = (H_1 - F)/S + 1$ and $D_2 = D_1$;
- Introduce zero learnable parameters;

3.3.2.3 Fully Conected Layers

Up until now no classification has actually been done in the previous layers, only feature extraction and size reductions occurred. Fully Connected layers are the ones responsible for the actual classification part of the network. As the name suggests, the neurons in these layers are connected to all neurons in the previous layer, and since these layers introduce nothing new in terms of operation from the hidden layers of the regular "plain vanilla" Neural Networks explained earlier in this chapter, in fact they are the exact same type of layers, not much will be added about these.

Fully Connected layers are used at the end of CNNs, after all feature extraction and size reduction is done. The output of the last feature extraction layer used (either Convolutional or Pooling) is flattened to a vector of neurons, which will serve as input to the first Fully Connected layer in the network, as seen in Figure 3.24. Keep in mind that the

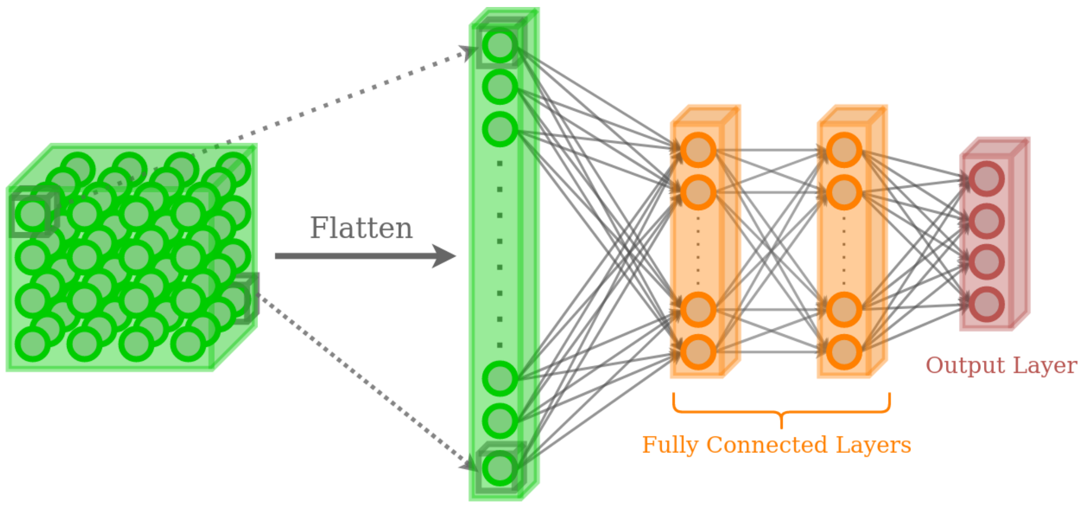


Figure 3.24: Fully Connected layers are placed in the end of CNNs. The last layer from the feature extraction part of the network is flattened into a vector and neurons in the fully connected layers connect to all neurons in the layer before them.

output of the previous layer should hold the activation maps of high-level features from the input image, and with this, the fully connected layers can learn which of these high level features most strongly correlate with a particular class.

Usually not many Fully Connected layers are used, not even in very deep CNNs, with the most notable architectures using a maximum of two or three such layers. The output layer, which is itself a Fully connected layer, will contain as many neurons as classes in the dataset, and each neuron will contain a class probability of the corresponding class. To achieve this it is common in CNNs, and for multi-class classification in general, to use as the activation function of the output layer the Softmax function:

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{3.30}$$

where $\vec{z}$ is the vector of values for each class related neuron to input the activation function, and K is the number of classes. This function converts the vector of output real values to a vector of output real values that sum to 1, effectively converting the output to a probability distribution for all classes.

3.3.3 CNN architectures - Classification

Now that the main types of layers are explained it is time to present some notable architectures that became famous over the last decade given their remarkable achievements on image classification and overall innovation in the field.

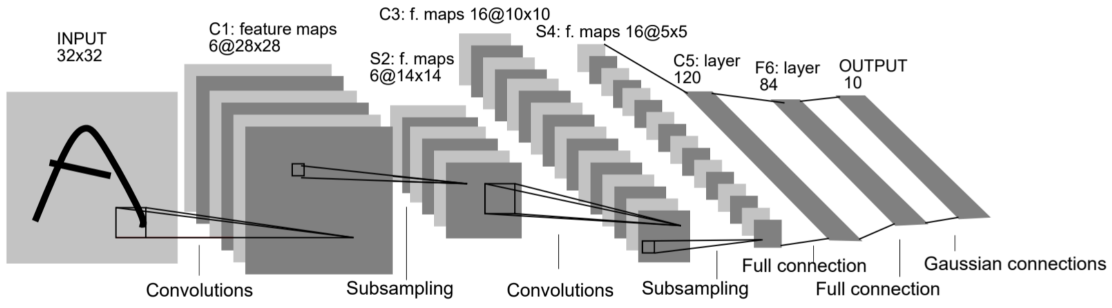


Figure 3.25: LeNet-5 architecture [44].

3.3.3.1 LeNet [44]

Presented in 1998 by LeCun et al. [44], LeNet was the first Convolutional Neural Network to be applied in practical applications, for handwritten digits recognition, and is the spark for the interest in Deep Learning in research. From three architectures presented in the original paper, the most successful one was LeNet-5 (Figure 3.25), that receives as inputs 32x32x1 images, introduced a total of 60850 parameters, and consisted of seven layers:

1. C1: Convolutional Layer with $K = 6$, $F = 5$, $P = 0$ and $S = 1$;
2. S2: Average Pooling Layer with $F = 2$ and $S = 2$;
3. C3: Convolutional Layer with $K = 16$, $F = 5$ $P = 0$ and $S = 1$;
4. S4: Average Pooling Layer with $F = 2$ and $S = 2$;
5. F5: Fully Connected Layer with 140 neurons;
6. F6: Fully Connected Layer with 84 neurons;
7. F7: Fully Connected Layer with 10 neurons;

Although achieving great results for the task it was given, classifying small black and white digits, the network had some limitations and design choices that have now been dropped from recent architectures. First of all the network was not very deep, which is not suitable for more complex datasets, as it used few filters per layer. It also used average pooling for subsampling, which nowadays is rarely used, with max pooling being now the preffered choice as it offers more rapid convergence since it returns larger gradients during backpropagation. Not only that, but the pooling method used also introduced learning weights and biases, applied to the results of the pooling itself. And lastly it used a scaled hyperbolic tangent function as activation functions, which nowadays has been replaced mostly by ReLU and ReLU variants.

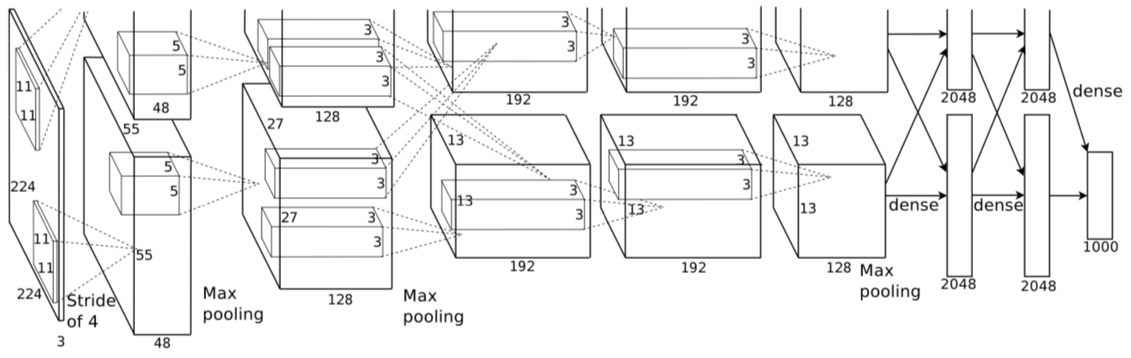


Figure 3.26: AlexNet architecture [30]. Here one GPU is training the neurons related to the top part of the image and the other is training the neurons related to the bottom part of the image simultaneously.

3.3.3.2 AlexNet [30]

Developed by Alex Krizhevsky et al. [30], AlexNet was the first CNN to win the ILSVRC (ImageNet Large Scale Visual Recognition Challenge) [45] in 2012, with such a large margin over the runner-up, a top 5 error of 15.3% for AlexNet compared to 26.11% error for the second place, that after it non-neural models became practically obsolete. For this reason it became one of the most influential architectures in Deep Learning. Its architecture was similar to LeNet-5, but deeper both in number of layers as in number of filters per layer, with 5 Convolutional Layers and 3 Fully Connected Layers, with some max pooling applied after the first, second and last Conv Layer, as seen in Figure 3.26. It also applied stacks of Convolutional Layers, when before it it was common to always follow a Convolutional Layer with a Pooling Layer.

Some of the reasons that made AlexNet stand out were: the use of ReLU as activation function, which led to faster training times when compared to similar networks using hyperbolic tangent; use of multiple GPUs during training, with one GPU training half the neurons and another GPU training the other half simultaneously, which also led to faster training times; the use of overlapping pooling, i.e. pooling with strides that overlap between subsequent positions of the filter, leading to an error reduction of 0.5%.

As for the number of trainable parameters, the network had around 60 million, which makes it prone to overfitting. To avoid it the authors used some data augmentation methods, generating new images in the dataset with translations and horizontal reflections of the original images, and used a technique during training called dropout, that consists of "turning off" a set of random neurons with a predefined probability at each training iteration. This provides robustness of the learned features, at the cost of increasing training time.

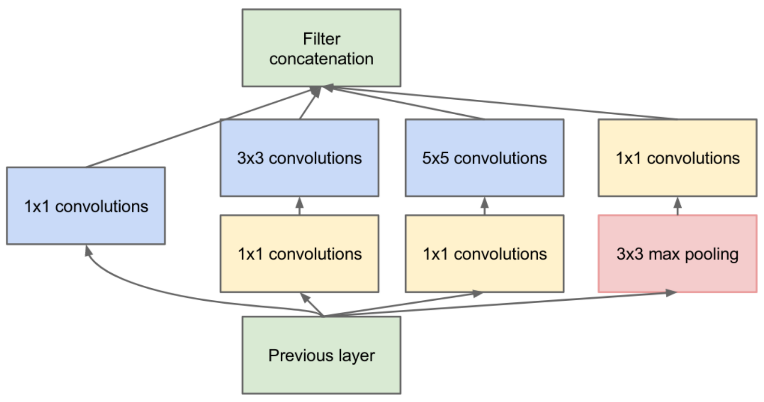


Figure 3.27: Inception module proposed by Szegedy et al. [46].

3.3.3.3 Inception and GoogLeNet [46]

Szegedy et al. proposed a new set of novel concepts to the way layers in a CNN could be deployed. They suggested a module that would apply Convolutional Layers of different filter sizes, 1x1, 3x3 and 5x5, as well as a max pooling in parallel, i.e. receiving as their input the same layer, and then concatenate all outputs into the same output volume along depth, effectively growing the network not only in depth as was usual in previous architectures, but also in width. The idea behind this was to be able to detect more features of different dimensions in a layer by applying different operations to it, as opposed to previous architectures that would only apply one type of operation or filter size per layer, possibly dismissing features that couldn't be captured by it. Another idea in the proposed module is to place 1x1 convolutions before the 3x3 and 5x5 and after the pooling layer before concatenating the results, in order to reduce the number of parameters in the network, that would otherwise become too large to be computationally feasible. These modules were named Inception modules, which can be seen in Figure 3.27, and would be stacked on top of similar modules to increase the networks depth and therefore the complexity of features detected. The networks developed using these modules were named Inception Networks, with Inception-V1 being the first, also known as GoogLeNet.

GoogLeNet, named in homage to LeNet, won the ILSVRC competition in 2014 with an amazing result of 6.67% top-5 error rate, a 56.5% relative reduction compared to AlexNet's result in 2012. This result was so remarkable that the organisers of the challenge had to evaluate human level performance to compare, with the human expert achieving a top-5 error rate of 5.1% after a few days of training. GoogleNet's architecture with the Inception module amounted to 27 layers, at an incredibly deep implementation as seen in Figure 3.28, but nonetheless having a reduced number of parameters, especially when compared to networks like AlexNet: only around 4 million parameters, against AlexNet's over 60 million.

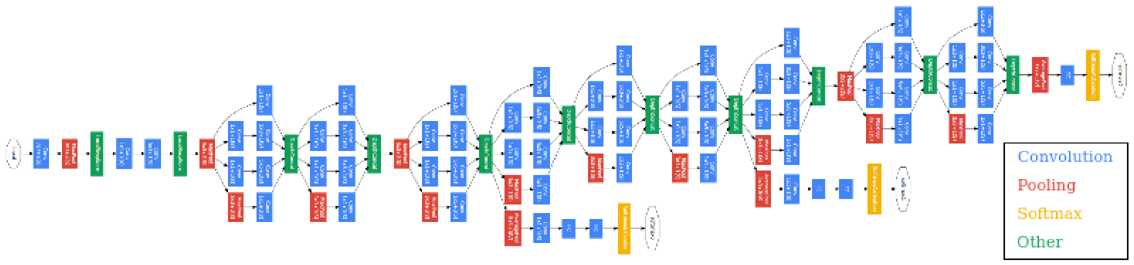


Figure 3.28: GoogLeNet architecture [46].

3.3.3.4 VGGNet [31]

Karen Simonyan and Andrew Zisserman from the Visual Geometry Group at the University of Oxford, participated in ILSVRC 2014 with a network architecture that became known as VGGNet [31], being runner-ups to GoogLeNet in the classification task. Highly influential in the world of Deep Learning, it shown the community that the depth of a network is critical for good performance, fueling the research in deeper CNNs. It also changed the common practice of applying bigger filter sizes in the first layers, by using only 3x3 convolutions along the network, showing that 3*3 convolutions stacked together can replicate bigger filter sizes and with more non-linearities present in between them. The specific architecture presented in ILSVRC is known as VGG-16, and displays an incredibly uniform architecture, by applying only 3x3 convolutions and 2x2 poolings, with the input's dimensions in width and height being halved as the network gets deeper, while the number of filters doubles, as can be seen in Figure 3.29.

Despite being runner-ups to GoogLeNet in the classification task, VGG-16 actually got better performance when applying only a single network (GoogLeNet's winning participation applied 7 networks), with a top-5 error of 7% against the 7.9% of a single GoogLeNet, and when using two networks the resutls came really close to GoogLeNet's winning participation, with a top-5 error of 6.8% against GoogLeNet's 6.7%. It also won the competition's object localisation task with a top-5 error of 25.3%, ahead of GoogLeNet's 26.7% result. In this task the last Fully Connected layer predicts the objects' bounding box location instead of the class scores.

The weight configuration of the trained VGGNet was released to the public, and the model has been used in many applications as one of the preferred baseline feature extractors. Because of its homogeneous architecture and great accuracy results, VGGNet is still used to introduce CNNs to newcomers, and real with its major drawback being its computational demands. Introducing some astonishing 138 million parameters, which results in really slow training sessions, and make its deployment less practical for real time applications.

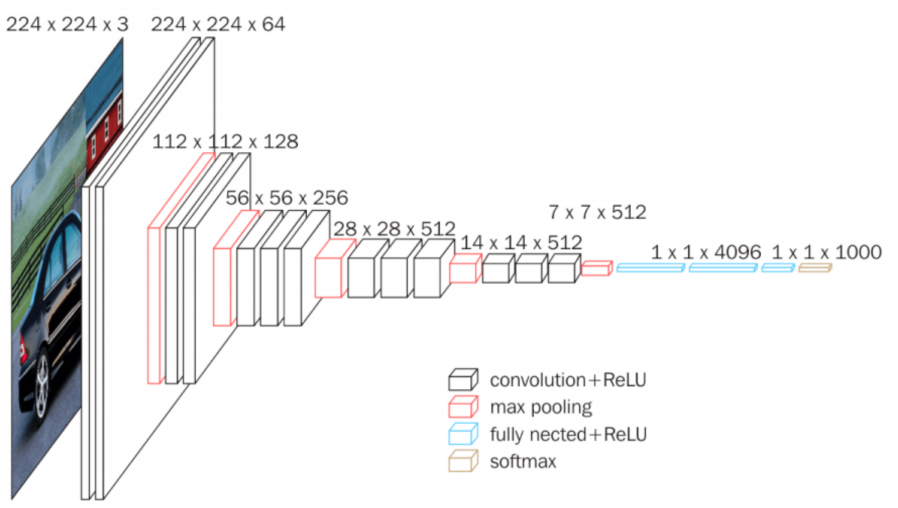


Figure 3.29: VGG-16 architecture (from [47]).

3.3.3.5 ResNet [48]

As mentioned earlier, as CNNs get deeper the gradients start to get smaller and smaller during backpropagation, as they reach shallower layers, in a problem known as vanishing gradient. However, VGGNet shown that the performance of a network is critically dependent on its depth. To be able to increase a network's depth without dealing with the vanishing gradient problem, He et al. introduced a new concept known as residual or skip connections which would create alternative paths for the gradients to reach a previous layer, by skipping layers in between them, as can be seen in Figure 3.30.

With such connections, He et al. were able to create and train incredibly deep networks known as Residual Networks, or ResNets, with the example of a particular model composed of 152 layer network that still has lower complexity than VGGNet. With these networks they won the ILSVRC in 2015 with a top-5 error rate of 3.57%, beating human level performance for the same dataset. Given this success, researchers started to adopt this type of connections in modern CNN architectures,being now a common practice.

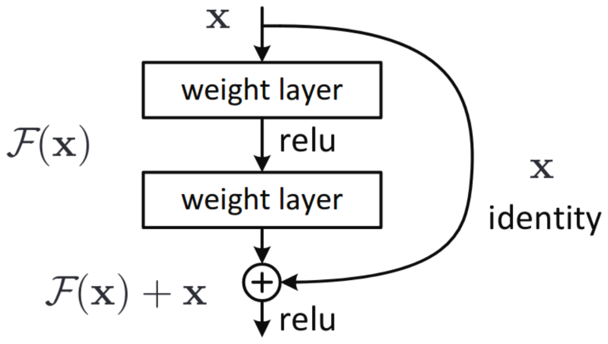


Figure 3.30: Residual connections block introduced by He et al. (from [48]).

3.3.3.6 MobileNet [49]

MobileNet is a CNN architecture especially designed with smaller, less computationally powerful devices in mind, such as mobile phones or Single Board Computers (SBC), in order to provide these devices the capability of using Deep Learning applications. Developed by Andrew G. Howard et al. [49], it introduced the concept of separable convolutions, that instead of applying K 3D convolutions across the layer depth, splits the task in two different convolutions: a depth-wise convolution that collects spatial information for each input channel, and a point-wise convolution that collects the interactions among the various channels. This distinction can be seen in Figure 3.31.

By applying separable convolutions the number of multiplications drops significantly. Considering D the dimension of the output, N the number of input channels, F the filter size and K the number of filters, the number of multiplications M taking place in a conventional convolution is given by:

$$M = F^2 \times N \times D^2 \times K \tag{3.31}$$

while for the separable convolution M is given by:

$$M = (F^2 \times D^2 \times N) + (N \times D^2 \times K) \tag{3.32}$$

For the example in Figure 3.31, this results in 86400 multiplications for the conventional convolution, while for the separate convolution it amounts to 10275. By applying this type of convolution, MobileNet can highly reduce the computational cost required, without hurting its accuracy significantly

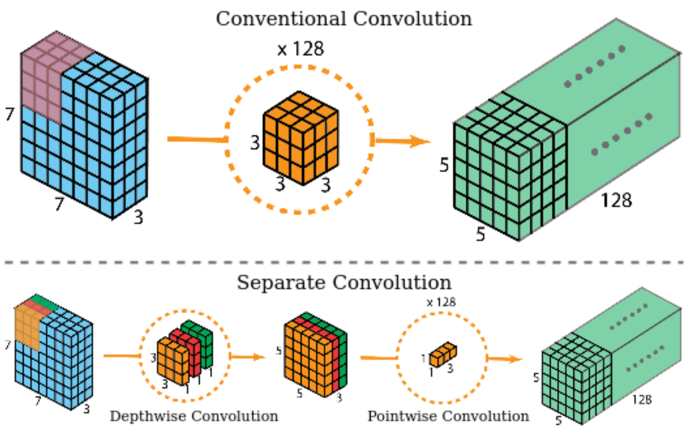


Figure 3.31: Conventional convolution and separate convolution (from [50]).

3.3.4 CNN architectures - Detection

The network architectures presented so far were designed with the sole purpose of image classification in mind, trying to label an image with a single class, no matter the number of objects in the image, or the positions of the objects in it, thus being, at a first glance, of no use for the object detection problem. However, there are other architectures and methodologies that focus on the detection of the objects in the image, as well as their classification.

3.3.4.1 You Only Look Once (YOLO) [51]

You Only Look Once (YOLO) is a milestone in the object detection field. As opposed to other methods that existed at the time of its release, like the R-CNN (Region-Based Convolutional Neural Networks) that split the object detection problem in two separate threads, a first step for region proposing and another for proposed region classification, YOLO was able to perform both tasks in a single step, hence its name. Presented in 2016 by Joseph Redmon et al. [51], YOLO divides the input image into an $S \times S$ grid, with each grid cell predicting a total of B bounding boxes, each with 5 predicted values (x, y, width, height and confidence), and a probability C for each class in the dataset, resulting in an output tensor with dimensions $S \times S \times (B * 5 + C)$. This process is visible in Figure 3.32.

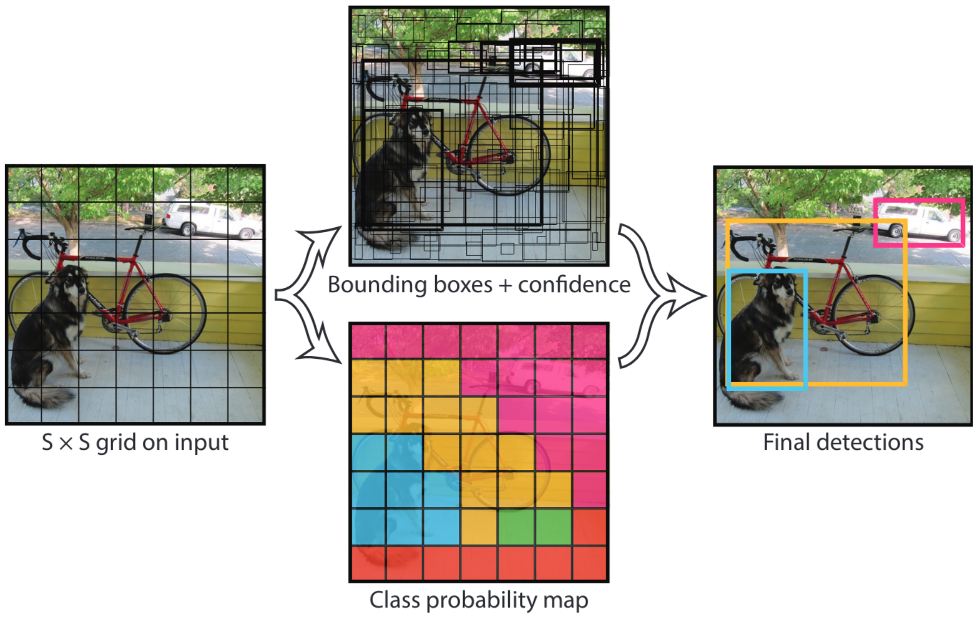


Figure 3.32: The YOLO model. In a single step YOLO can predict bounding boxes and class probabilities from an image, by dividing it in an $S \times S$ grid, with each grid cell predicting B bounding boxes and their confidences, as well as class probabilities (from [51]).

As for the network architecture YOLO was inspired by GoogLeNet, simply using 1×1 reduction layers followed by 3×3 convolutional layers, instead of the inception modules of GoogLeNet. Two architectures were proposed, one with 24 convolutional layers followed by 2 fully connected layers for class and bounding box prediction, and another known as Tiny YOLO which only contained 9 convolutional layers, for even faster object detection, at an accuracy cost. YOLO achieved an mAP (mean Average Precision) of 63.4% on the VOC (Visual Object Classes) 2007 dataset running at 45 FPS on an Nvidia Titan X GPU. Tiny YOLO achieved an mAP of 52.7% on the same dataset, but performing at 155 FPS.

Despite its success at the time, YOLO had some limitations, such as difficulties in detecting small objects, or objects positioned too close to others, and difficulties in detecting objects with unusual aspect ratios. To overcome these problems new versions have been introduced throughout the years, with YOLOv2 [52] released in 2017 introducing significant improvements on anchor boxes and higher resolution, achieving 76.8 mAP at 67 FPS, and 78.6 mAP at 40 FPS on VOC 2007, and YOLOv3 [53] released in 2018 that improved performance on small objects.

3.3.4.2 Single Shot Multibox Detector (SSD) [54]

The Single Shot Multibox Detector is an object detector based on Convolutional Neural Networks. Presented in 2016 by W. Liu et al. [54], SSD starts with a base CNN architecture for feature map extraction, followed by convolutional filters applied to the feature maps to predict bounding box locations and classifications at different scales, effectively locating objects of different dimensions in the image, and providing the best classification for each predicted bounding box. In the original paper the base network used was VGG-16, to which the fully connected layers are removed, working therefore only as a feature extractor. After the base network a set of convolutional layers are added with decreasing dimensions from one to the next to allow detection of objects of different scales. This architecture can be seen in Figure 3.33.

The resulting feature maps are divided into grids, and a set of predefined bounding

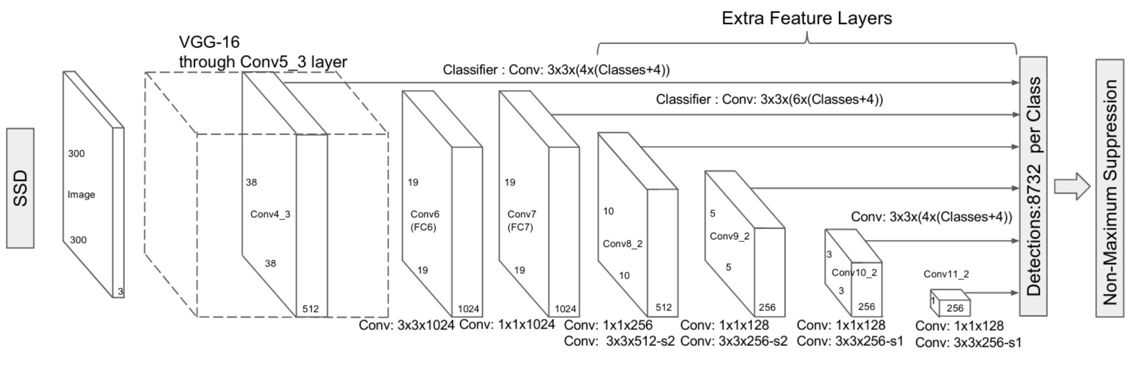


Figure 3.33: SSD architecture with VGG-16 as base network (from [54]).

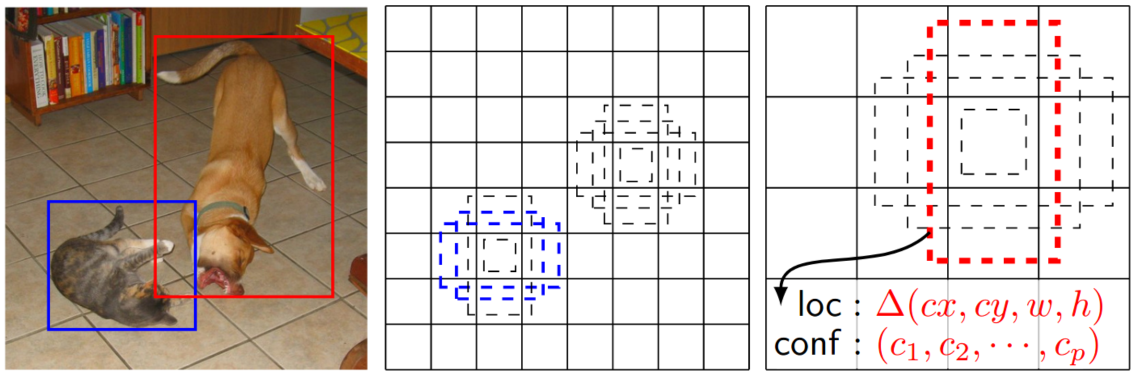

loc : $\Delta(cx, cy, w, h)$
conf : $(c_1, c_2, \cdots, c_p)$

Figure 3.34: SSD methodology. Left: the only thing needed for training of the network is the images and ground truth boxes. Centre: Feature maps are divided into grid cells and each cell is equipped with a set of predefined bounding boxes for classification, each predicting a confidence for each class in the dataset. Right: feature maps at different scales allow detection of objects of different dimensions, with adjustments to the predicted boxes taking place during training to fit the ground truth better (from [54]).

boxes with different sizes and aspect ratios are assigned to each grid cell. Then the network predicts a score for each class in the dataset for every bounding box, adjusting each box shape as training proceeds, in order to fit the ground truth boxes better. If nothing is detected in the grid cell, it is considered background. Given the amount of prediction boxes this process generates, Non-Maximum Suppression is applied to eliminate redundant predictions. This process can be seen in Figure 3.34.

The network was able to achieve 74.3% mAP on the VOC2007 test set at 59 FPS on a Nvidia Titan X for 300x300 input images, and 76.9% mAP for 512x512 input images. Despite being implemented with VGG-16 in the original paper, the authors suggest that it should also achieve good results with other base architectures.

This page was intentionally left blank.

Chapter 4

System Architecture

In this chapter the decisions made to fulfil the project's objectives are introduced and explained, along with the high level hardware and software architecture of the proposed solution. After careful study of the state-of-the-art on zooplankton detection and classification, both in existing *in situ* plankton imaging systems that apply their methods directly, mainly for plankton detection, and in independent approaches that work over already existing datasets focusing on ways to perform better classifications, and with a new understanding of the concepts behind Convolutional Neural Networks, it was possible to envision a solution that would satisfy the project's objectives proposed in Chapter 1.

4.1 Solution Description

The presented solution aims for both zooplankton detection and classification *in situ* and in real-time, in a way that can be easily deployed in MarinEye's plankton imaging system. Given these requirements the choice fell for a conjugation of CNNs that address both problems in an effective way. For this, a decision for the networks architectures and deep learning engine for the system was made in a way that would guarantee system portability and reduced cost while maintaining a good performance. To make such decisions it was first necessary to know the specifications of MarinEye's imaging system.

4.1.1 MarinEye's Imaging System

For plankton imaging, MarinEye is equipped with an IDS UI-3590CP-C-HQ USB 3.0 camera with a resolution of 18MP and a pixel size of 1.25 μm. Coupled is a DTCM230-36 telecentric lens with a working distance of 110 $\pm$ 2 mm, a depth of field (DOF) of $\pm$ 4.3 mm at a F11 aperture and a magnification of 0.317x. An external LED illumination strobe, placed directly in front of the lens, provides scene illumination. To acquire and process the images, and to communicate with the other MarinEye subsystems, the imaging system is equipped with an Odroid-XU4 with two CPU, an ARM Cortex-A15 2 GHz Quadcore and

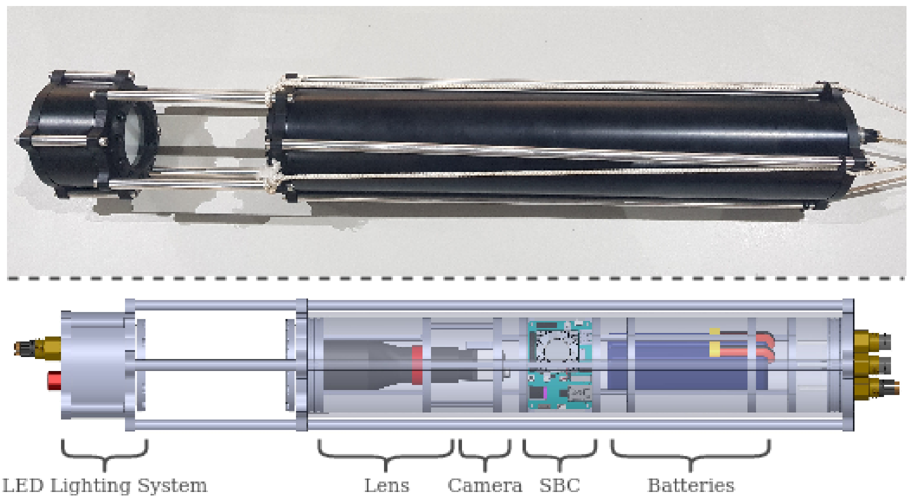


Figure 4.1: Top: MarinEye's plankton imaging system. Bottom: System's different components (adapted from [55]).

a Cortex-A7 1.3 GHz Quadcore. The boards GPU is a Mali-T628 MP6, and it has 2 GB of RAM. The operating system installed is Ubuntu Mate 16.04. As for the software, the system uses ROS[1] (Robot Operating System) for control, communication and data transfer between its components. ROS is a free and open source software framework for robotic applications. It's comprised of a variety of tools and libraries that creates a hardware abstraction layer which allows easy communication between different elements running in the system, from sensors to actuators in a simple methodology. It also offers libraries and tools for simulation and visualization. Processes running in ROS are called nodes, and different nodes communicate with each other in a publisher-subscriber fashion via messages published under specific topics. This type of mechanism allows easy integration of new applications in the system, since nodes can publish their own data for it to be easily available for any other node in the system that subscribes to it.

MarinEye's plankton imaging system, fully developed in the laboratory, can be seen in Figure 4.1, and more specifications of the whole system can be found in [55].

4.1.2 CNN Architectures and Engine

From the research done in Chapter 2 the decision on the classification algorithm to use fell on Convolutional Neural Networks, as they dominate the state-of-the-art in image classification, whatever the object for classification is, a scenario also observed in plankton classification, with every recent approach resorting to it with good results. For the detection of zooplankton the presented solution proposes an original approach to the problem, by also using a CNN for the task, as opposed to other *in situ* plankton imaging systems

[1]https://www.ros.org/

that make use of traditional computer vision and image processing algorithms. The use of CNNs for zooplankton detection *in situ* introduces innovation in the field, with MarinEye being the first of such systems to follow this approach, as far as the conducted research allowed to infer.

The idea to use two different models, instead of leaving classification and detection for the same network, was manifold. First, because of the inexistence of publicly available zooplankton detection datasets. Second, inspired by other plankton imaging systems that provide a broader classification in the detection, followed by a more specific classification. This way the detection CNN could give basic classification for non-class specific tasks, such as plankton counting and biomass estimation, or to objects that the classification network is unable to provide a classification with enough confidence, i.e. with low probability of the inferred class.

Despite the great performance of CNNs in image classification accuracy-wise, they are still computationally demanding algorithms, being too demanding to deploy in MarinEye's imaging system SBC unit to take care of all the processing tasks related to them, while still having to control every other component in the system. With this configuration detection and classification couldn't be performed in real-time. By the time this project was being planned Intel® launched a novel device called Movidius™ Neural Compute Stick (NCS)[2], which is a USB 3.0 stick especially designed for deep learning applications on devices with low computational power, allowing to run CNN inferences in real-time, visible in Figure 4.2. Alternatively, a decision could be made to change the imaging system's computational unit altogether for a more powerful device prepared for more demanding applications, perhaps with a powerful GPU, such as the Nvidia Jetson TX1 Developer Kit [3].

The choice for the Intel® Movidius™ NCS as the system's deep learning engine fell on its size, portability and low-cost characteristics, making it suitable for the need of an easily deployable and high performance solution. With this device not only would it not be necessary to make changes on the existing plankton imaging system, but it is also possible to easily deploy deep learning applications in any other system from the laboratory that

Figure 4.2: Movidius™ Neural Compute Stick.

[2]https://www.intel.com/content/www/us/en/developer/articles/technical/intel-movidius-neural-compute-stick.html

[3]https://developer.nvidia.com/embedded/jetson-tx1-developer-kit

could benefit from it, just by plugging in the device in an available USB port.

The NCS is powered by the low power high performance Intel® Movidius™ Myriad™ 2 Vision Processing Unit (VPU). It is also equipped with 4Gbits of LPDDR3 DRAM, imaging and vision accelerators, and 12 vector processors called SHAVE (Streaming Hybrid Architecture Vector Engine), which are used to accelerate neural networks by running parts of the network in parallel. A Software Development Kit (SDK), called NCSDK, containing a set of software tools, and an API for both C and Python programming languages, compatible with deep learning frameworks Caffe[4] and Tensorflow[5], allow to profile, tune, and deploy a set of Convolutional Neural Networks on low-power applications requiring real-time inference, when combined with the NCS.

4.2 High-Level System Architecture

The high-level hardware architecture for the proposed solution is pretty straight forward. Two Movidius™ NCS were used, one for each network, connected to the system's processing board, the Odroid-XU4, by the USB 3.0 port. The board controls all devices, the camera, the lighting system and the NCSs, and sends the images captured by the camera to the NCS, receiving the inference results from it. A representation of the high-level architecture is seen in Figure 4.3.

The software developed for the system is implemented as a ROS node that subscribes to the raw image topic from MarinEye's camera, and publishes both NCS inference results, so that they can be used for any desired purpose by an user application node (e.g. zooplankton counting and biomass estimation).

First the node makes the necessary NCS initialization, loading in them the network related files. For this, the node makes use of the Movidius™ corresponding API functions.

Once both NCSs are correctly initialized, the node subscribes to the images topic to feed them to the detection CNN's corresponding NCS and obtains its inference results. These results consist of a list of zooplankton objects, ROI location in image and probability of object in ROI being zooplankton for every detected bounding box.

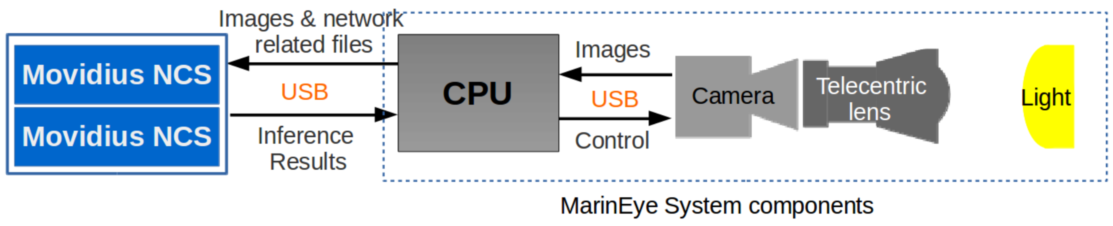


Figure 4.3: High level hardware architecture.

[4]http://caffe.berkeleyvision.org/
[5]https://www.tensorflow.org/

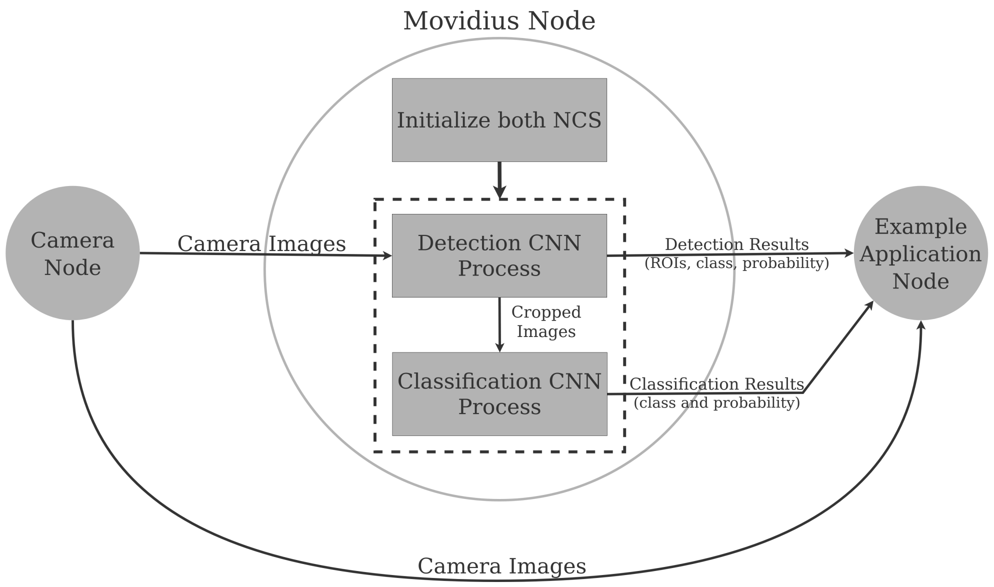


Figure 4.4: Software Architecture.

From the previous results, the images to feed the NCS related to the classification CNN are cropped from the original image, creating new images with single organisms to be classified. The classification results from these cropped areas are added to the detection results if they return a confidence greater than a specified threshold, otherwise the broader classification from the detection network is kept. The results are then published in a new topic. In figure 4.4 the software high level architecture is depicted.

By implementing the software as a ROS node, it becomes even easier to deploy the plankton detection and classification application to any other system, be it an updated version of MarinEye's imaging system, or to any other LSA robotic platform.

This page was intentionally left blank.

Chapter 5

Implementation

In this chapter the implementation details and methodology followed for each moment of the project, in order to fulfil the proposed objectives are shown. First the work developed in an initial iteration of this project is presented, from dataset creation and gathering to the experimental configuration used, followed by the respective results. Then the implemented improvements over this initial work are also reported, with focus on system upgrades and more methods for obtaining multiple metrics that could provide a better insight on the overall system performance.

5.1 First Experiments

Here an initial approach to the project is presented in the first experiments made. In this first iteration the CNN architectures to use in the project were chosen based solely on the study of the state-of-the-art and the system limitations. For detection an SSD implementation with MobileNet as base network was used, for a fast region of interest obtainment to feed to the classification CNN. By using an SSD model it would be possible to perform object detection, and since MobileNet was designed with less computationally capable devices in mind, it would be ideal for use as the base network in order to guarantee real-time execution.

For an accurate classification of the regions of interest provided by the previous network, Jialun Dai's ZooplanktoNet presented in Chapter 2 was used, given its author's claims of better performance when compared with other typical and popular architectures when accuracy, loss value, training time and model complexity are taken into consideration [28].

5.1.1 Datasets

The first step for training the networks is gathering the datasets, one for the detection network and another for the classification network.

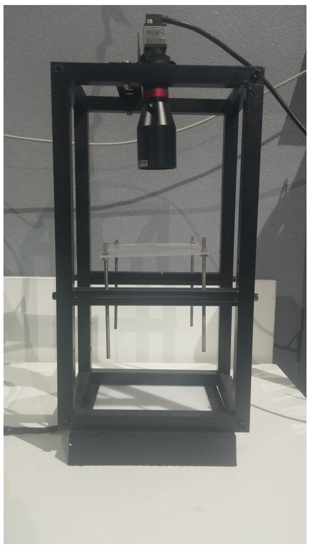


Figure 5.1: Experimental setup with MarinEye's imaging system components.

Given the inexistence of publicly available zooplankton datasets for detection or segmentation, it was necessary to create one from the start, from gathering the images, to manual labelling. To create the dataset an experimental setup was used, containing MarinEye's imaging system's camera and lens mounted in a structure with an acrylic plate between the camera and a light source, at the camera's focal distance, where a Petri dish containing zooplankton samples could be placed. This setup can be seen in Figure 5.1.

With this setup it was only necessary to obtain preserved zooplankton samples, which were provided by CIIMAR (*Centro Interdisciplinar de Investigação Marinha e Ambiental*). From this sample a total of 100 images were captured with two different light modes, dark-field and bright-field, in order to try to provide the system the flexibility to perform well under different conditions. Some of these images are visible in Figure 5.2.

After obtaining the images it was time to label each in a binary classification manner,

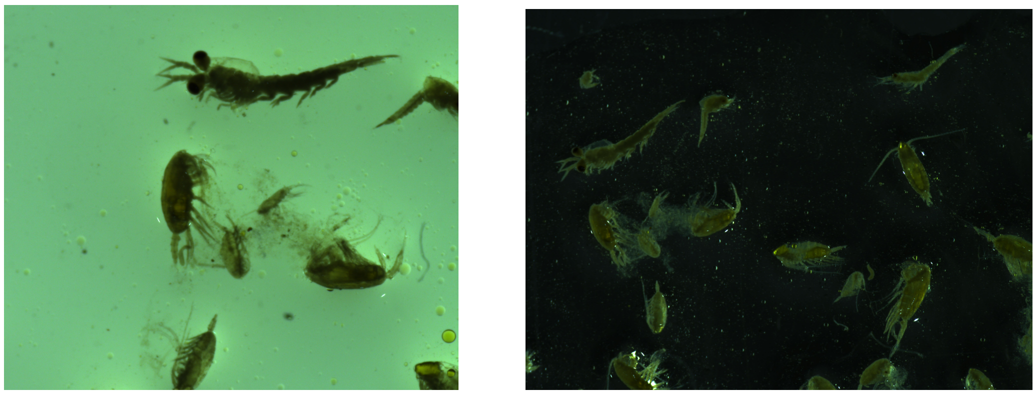


Figure 5.2: Example images obtained with the experimental setup under two different light modes, bright-field (left) and dark-field (right).

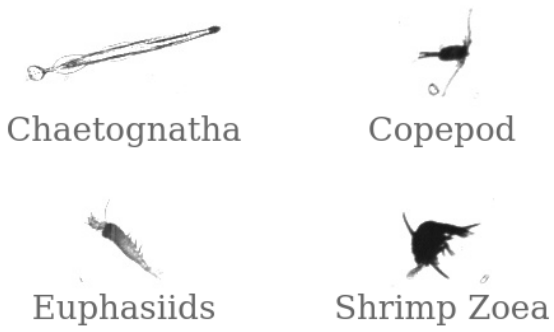


Figure 5.3: Example images from the Kaggle dataset.

i.e. annotating image by image the regions where there was zooplankton, and label it as such. For this a bounding box drawing script was used, that generated the image annotation files in PASCAL (**P**attern **A**nalysis, **S**tatistical modelling and **C**omput**A**tional **L**earning) VOC format. To reduce overfitting and to increase the number of images in the dataset, a data augmentation process was performed, using a Python script developed for processing the image transformations, and consequently, generating the new bounding boxes and annotation files relative to the transformed images. These transformations consisted of:

- Rotations of 45º steps clockwise;
- Vertical and horizontal flip;
- Gaussian blur with three different values;
- Re-scaling by 0.5x and 2x;

The final dataset was then composed of 960 images.

For the classification network the Kaggle dataset [56] was used. Kaggle is a plankton imagery dataset used for the National Data Science Bowl competition, gathered by the In Situ Ichthyoplankton Imaging System (ISIIS) [57]. This dataset consists of 121 plankton classes, with classes ranging from 108 to 1979 images per class. Some images from this dataset are seen in Figure 5.3.

A data augmentation process similar to the one applied to the detection dataset was also implemented. After data augmentation both datasets were split in two, one for training and another for testing, in a ratio of 90%-10%, being then converted to a LMDB (Lightning Memory-Mapped Database) database for network training.

5.1.2 Experimental Configuration

Both networks were implemented using the Caffe framework, and trained using a Nvidia® Jetson TX2 platform based on a Nvidia Pascal GPU with 256 Nvidia CUDA cores and 8 Gb of RAM memory. The training stage was performed until the loss stabilized, generating the networks trained models. To validate that with Movidius™NCS is possible to have a portable and easily deployable low cost detection and classification system without compromising the performance of the networks accuracy, while being able to achieve real-time inference, two different ROS nodes were tested, where the functionality is the same, but one uses the NCS to do the inference on an input video, while the other uses the Caffe API installed on the machines native system. This test was executed in a laptop powered by an Intel® i5-6300HQ, Quad-Core, 2.30GHz running Ubuntu 16.04 operating system, and on MarinEye's Odroid-XU4. All systems had ROS Kinetic distribution. Performance results are depicted in Section 5.1.3.3

Given the differences in images from the classification dataset to the ones obtained from the experimental setup, some image pre-processing is applied on the detected ROIs, consisting of a conversion to gray scale followed by an adaptive threshold on pixel values, in order to improve classification accuracy of the full system.

5.1.3 Results on the First Experiments

Initially, the performance of the two networks was verified separately when applied to the respective test sets, with an overall system evaluation done afterwards.

5.1.3.1 Detection

MobileNet-SSD trained during approximately three days on the Jetson TX2. In figure 5.4 it's possible to see detection results on images from the detection test set, both bright- and dark-field.

The confidence on the detections in the presented images is very high on both the bright-field and the dark-field scenarios. However the network precision can't be judged by image observation alone and confidence of the bounding boxes, but on error data

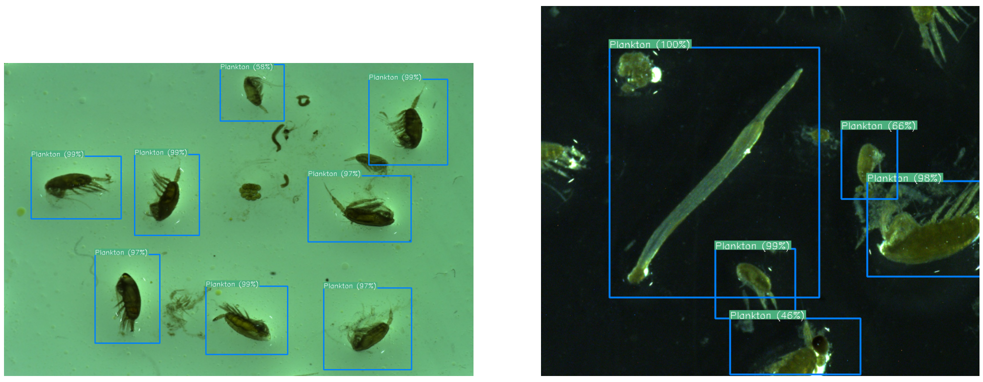


Figure 5.4: Detection results from the first iteration, bright-field (left) and dark-field (right).

analysis. The first values to obtain to evaluate the network efficiency are the precision and recall for different confidence threshold values. Confidence threshold is the minimum confidence value accepted as a true positive from the inference result. Precision and recall are obtained from the number of true positives (objects that the network correctly predicts as zooplankton), false positives (objects that the network wrongly predicts as zooplankton) and false negatives (objects that the network should have predicted as zooplankton but didn't), as given by Equations 5.1 and 5.2:

$$Precision = \frac{TP}{TP + FP} \tag{5.1}$$

$$Recall = \frac{TP}{TP + FN} \tag{5.2}$$

In Table 5.1 values of TP, FP, FN, precision and recall on the test set images are shown for different threshold values. The bold rows in the table show the threshold values that provide the maximum precision and recall.

Table 5.1: Detection error analysis

Threshold(%)	TP	FP	FN	Precision(%)	Recall(%)
25	**492**	**126**	**167**	**79.61**	**74.66**
45	467	113	192	80.52	70.86
65	433	94	226	82.16	67.71
87	**367**	**59**	**292**	**86.15**	**55.69**
95	292	53	367	84.64	44.31

In Figure 5.5 the precision-recall curve is presented, and in Figure 5.6 both precision and recall are shown in relation to the threshold value.

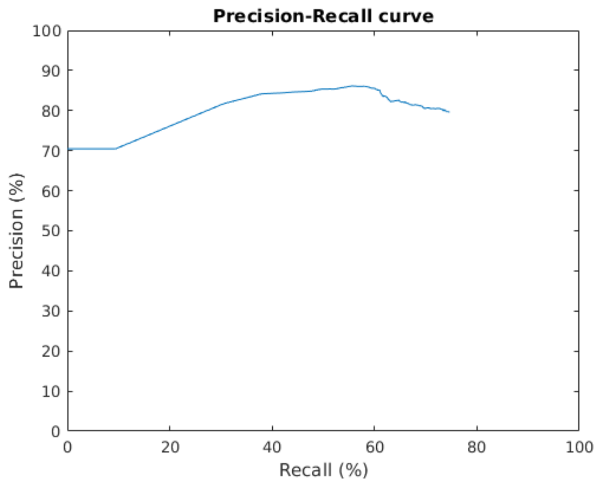


Figure 5.5: Precision-Recall curve for MobileNet-SSD

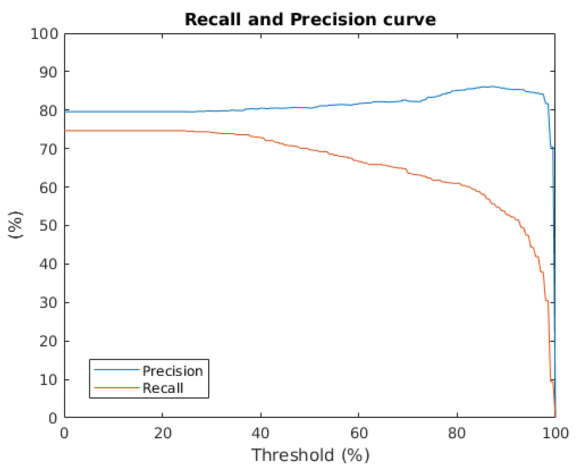


Figure 5.6: Precision and Recall curves for MobileNet-SSD.

For a better figure 5.5 evaluation, an analysis on figure 5.6 is proposed. The high precision for low threshold values in figure 5.6 is explained by the lack of non-zooplankton objects in the dataset images and the cause for it to only reach as much as 86.15% is explained by the high confidence the inference assigns to the false positives (above 90%). From this figure analysis it's also possible to see that the recall values start from around 75% even for a threshold value of 0%, instead of starting from close to 100% as it would be expected, for every zooplankton would have been detected for such low threshold value, no matter the confidence assigned, leading to the non existence of false negatives. This suggests that the training was not good enough, probably due to its small dataset, short training time and lack of non-zooplakton objects. A common metric to evaluate detection algorithms is the mean of Average Precisions, or mAP. As the name suggests, mAP is the mean of the Average Precisions, which itself is a metric that evaluates a network performance based on precision and recall values, given by the area under the precision-recall curve. When there are multiple classes in the dataset, the mean of all classes AP is obtained, hence mAP. When there is only one class in the dataset, as is the case, mAP and AP have the same value. In this experiment the network rached an mAP value of 75%.

5.1.3.2 Classification

ZooplanktoNet training ran for two days and obtained an accuracy of 83%. This value is far from the results on the original paper, granted that for this the test conditions were different, since the dataset was a completely different one, with more than 100 classes to train, instead of the 13 classes dataset used in the networks original paper, and more imbalance of data between classes.

In Fig 5.7 a graph of the accuracy over the number of training iterations for the classification network is shown. Here it is possible to see that the set training time was

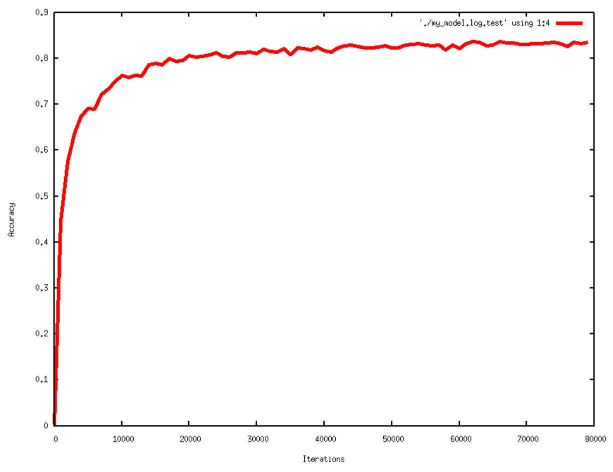


Figure 5.7: Accuracy over iterations from ZooplanktoNet

sufficient, as test accuracy stabilized around 83% and the network did not overfit, as if that was the case, validation accuracy would start to decrease.

In Figure 5.8 it is possible to see classification results from ZooplanktoNet on the extracted ROIs from the detection network, alongside images from the classification dataset corresponding to the class they were classified as. Here it is possible to see that even though the network was trained with a dataset vastly different from the images cropped from the detection dataset, it was still able to generalize well enough for this new and different data, which is an indication that the learned features may indeed be intrinsic to the specific classes, and not just an overfit phenomena over the dataset it was trained with.

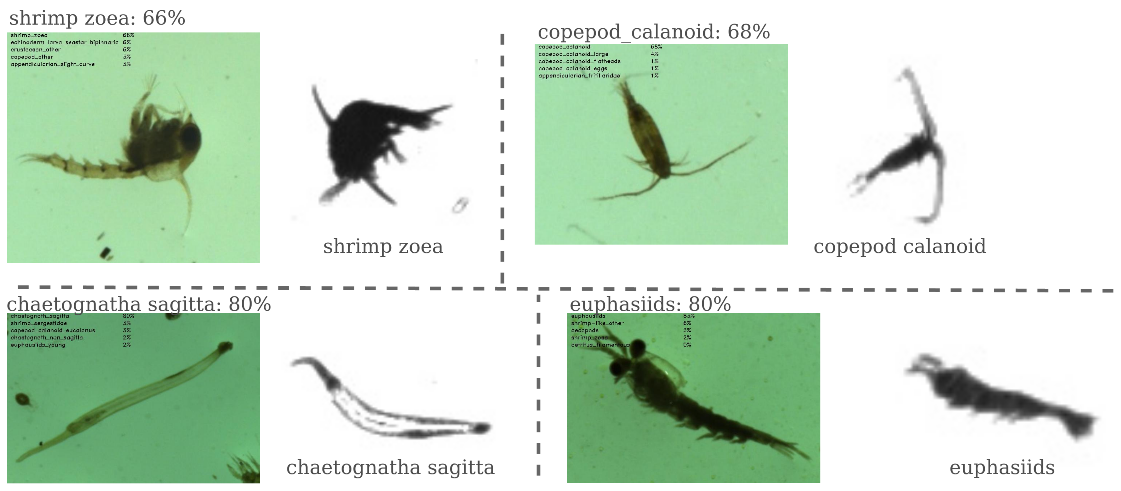


Figure 5.8: Classification results from ZooplanktoNet over cropped ROIs from the detection network, alongside images from the corresponding class from the classification dataset.

5.1.3.3 System evaluation

The evaluation of the combined system is based on the inference speed on both platforms, the Odroid XU4 and the laptop mentioned in Subsection 5.1.2. In table 5.2 it's possible to see the maximum and minimum frame rate when using the Caffe API on each platform's CPU and the developed software using the NCSs.

Table 5.2: Inference speed results

Platform	CPU (fps)	NCS (fps)
Laptop	0.1	1.3 to 9.7
Odroid-XU4	0.02	1.5 to 7

These results show that the use of the Movidius™NCS allowed to get a big improvement on the inference speed for the proposed frame resolution of 1920x1080 px. Increasing the image resolution, it was found that, although a larger sample is obtained, the accuracy of the detection network is very poor. On the other hand, decreasing the resolution allows a large increase in the detection precision. This is due to the fact that it is necessary to resize the images to be in accordance with the input expected by MobileNet-SSD, 300x300.

5.2 Improvements on Previous Work

Since the first iteration of the project some improvements have taken place, both to increase system performance and to keep it up to date with more recent methodologies, as well as to obtain important metrics for system evaluation.

5.2.1 Datasets

The first rectification done to the previous work was on the datasets. For classification a dataset obtained with the ZooScan system [27] and publicly available online was used for three main reasons: images in this dataset are more alike the images extracted by the detection network, which should in principle contribute to increase accuracy on the classification of the captured images; number of classes in dataset is smaller and more representative of the species also found in the images captured for the detection dataset, with the classes being a little more balanced in terms of number of images; it is the same dataset used in ZooplanktoNet's paper [28], allowing to try to reproduce the same methods, in an attempt to achieve similar results, and to compare it to other networks not mentioned in the original paper. This dataset contains 9460 grayscale images distributed by 13 different classes, with 9 classes of single zooplankton individuals, a class for multiple objects on the same image and three other classes, fibers, bubbles and non-bio. A data

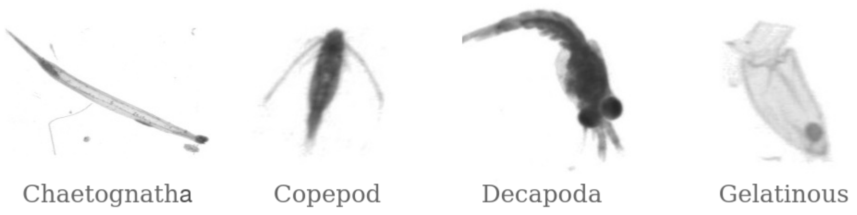


Figure 5.9: Example images from the ZooScan dataset.

augmentation process was also applied, this time following a similar approach to the one used in the ZooplanktoNet's paper, with similar rotations, translations, rescaling, shearing and flipping processes, with the addition of gaussian blur. Some images from this dataset can be found in Figure 5.9.

In the detection dataset some rectifications were also applied. First more images were obtained, since the original dataset was too small for Deep Learning projects, this time during several sessions of sampled plankton image collecting at CIIMAR, from which more than 5000 images were captured. In this new iteration only bright-field lightning was used, as this is the mode the MarinEye's plankton imaging system captures its images during operation. Images that contained artifacts and other non-zooplankton objects were also added to perform a more robust training. After a careful inspection of these images, the ones that were too blurred, too similar, that had no significant objects of interest or with too many beings too close together were eliminated, resulting in a total of 396 images. These were then labelled with the same image labelling script used in the first implementation. The data augmentation process was also improved, adding more transformations that could increase the system's immunity to different conditions of the environment, consisting of:

- Vertical and horizontal flip;
- Re-scaling by half, 512x512 and 300x300;
- Three rotations of 90^{o} steps;
- Changes in hue and saturation with two different values;
- Gaussian blur with three different values;
- Contrast normalization with two different values;
- Conversion to grayscale with two different values;
- Gaussian noise;
- "Salt and pepper" noise;

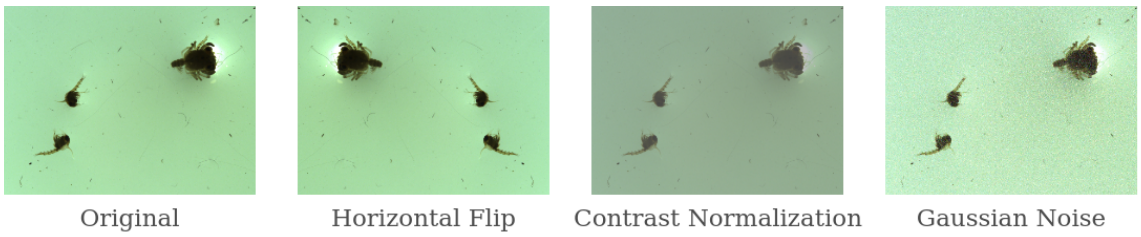


Figure 5.10: Examples of data augmentation.

The final dataset after data augmentation consisted of 7920 images. Some examples of augmented images from this dataset can be seen in Figure 5.10.

This time both datasets were split in training and test datasets by a ratio of 80%-20% respectively.

5.2.2 CNN training

Another improvement on the previous work was the introduction of new classification and detection networks, in order to evaluate the system with different CNN configurations. For classification, new networks were trained with the new dataset: GoogLeNet, Inception-V2, Resnet-18, VGG-16 and ZooplanktoNet. The training parameters for each network can be observed in Table 5.3.

Table 5.3: Classification networks training configuration

Network	Base lr	lr Policy	lr Parameters	Batch Size	Epochs	Training time
GoogLeNet	0.01	Poly	Power = 2	64	60	1h 28min
Inception-V2	0.05	Poly	Power = 2	32	60	2h 25min
Resnet-18	0.1	Poly	Power = 2	128	60	1h
VGG-16	0.005	Poly	Power = 2	32	26	3h 46min
ZooplanktoNet	0.01	Step	Gamma = 0.1	100	35	36min
			Stepsize = 5000			

The parameter "Base lr" is the starting learning rate for the Gradient Descent algorithm, "lr Policy" is the method of how this learning rate should decrease during training. With "Poly" the effective learning rate follows a polynomial decay to be zero by the last iteration, given by:

$$lr = lr_0 * (1 - \frac{i}{T_i})^{Power} \tag{5.3}$$

with lr_0 being the base learning rate, i being the current iteration and T_i the total number of iterations. "Power" is the parameter set to adjust the polynomial decay. The "Step" policy means that the learning rate should be reduced by a factor of "Gamma" after a given number of iterations, the "Stepsize".

For detection the Tiny-YOLO-V3 network was implemented for inputs of 608x608, with a batch size of 32, base learning rate of 0.02, with a "Step" policy with a Gamma of 0.1. This network was chosen given its good performance when both inference speed and precision are taken into account.

The classification networks were implemented in Caffe, and Tiny-YOLO-V3 was implemented in its official framework Darknet. The workstation used to train the networks contained two Nvidia GeForce RTX 2080 ti with 8Gb of memory each.

5.2.3 Software

Some time after the release of the Movidius™ NCS, Intel introduced support for the OpenVINO™ framework with this device, which supported more network architectures and frameworks than NCSDK, which eventually became deprecated. OpenVINO™ provides a unified methodology for many different computing architectures, from CPUs, GPUs, the Movidius™ NCS, and FPGAs (Field Programmable Gate Array). The framework provides two main components important for CNN applications, the Model Optimizer and the Inference Engine. The Model Optimizer allows conversion of network weight files from their native framework to an Intermediate Representation (IR) that is supported by OpenVINO. The Inference Engine provides an API for both Python and C++ to read the IR files and execute the models on the inference device.

The software developed for the zooplankton detection and classification was therefore updated to make use of OpenVINO's tools.

To evaluate how the detected ROI pre-processing step affects the classification accuracy five new small test datasets with five classes also found in the ZooScan dataset were created with these obtained images at varying values for the adaptive threshold technique applied, to evaluate the trained networks performance on them, with an example images of these datasets displayed in Figure 5.11. The same data augmentation method used in the ZooScan dataset was also used in these datasets.

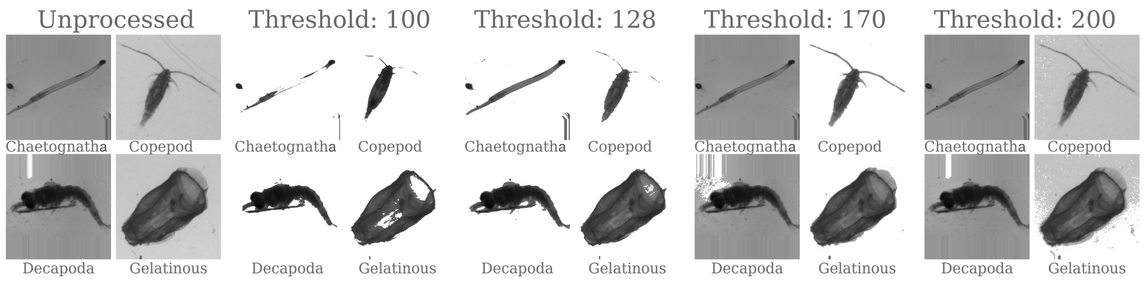


Figure 5.11: Example images of created datasets at various threshold values.

This page was intentionally left blank.

Chapter 6

Results

As mentioned in Chapter 5 a set of improvements to the overall system were introduced on all of the system components: from the CNNs and datasets tested, to the API used for the ROS node. In this chapter the results from these improvements are shown.

6.1 Detection

Training was executed for 15312 iterations, which is around 78 epochs for this dataset with the set batch size. The training loss per iteration graph of Tiny-Yolo-V3 can be observed in Figure 6.1. This figure suggests that training wasn't stopped too early, as the error seems to have started to stabilize around a value of 2, but could have been executed for some more time. Nonetheless it reached a maximum mAP of 95.54%, largely surpassing the value achieved by MobileNet-SSD in the first iteration of the project, in a far more complex and complete dataset less prone to overfitting.

In Figure 6.2 the precision and the recall curves over the confidence threshold are shown. In this figure the result looks much more like what would be expected from a well

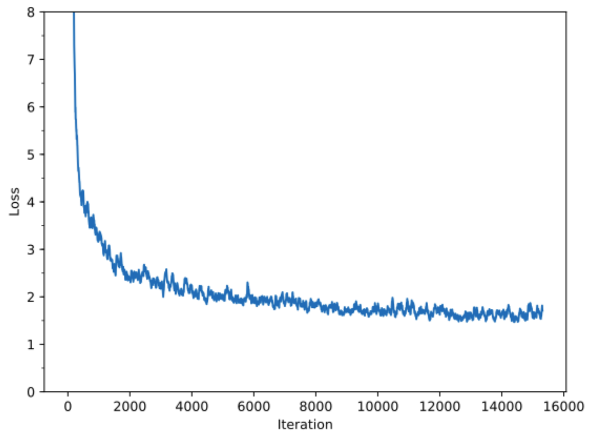


Figure 6.1: Tiny-YOLO-V3 training loss per iteration.

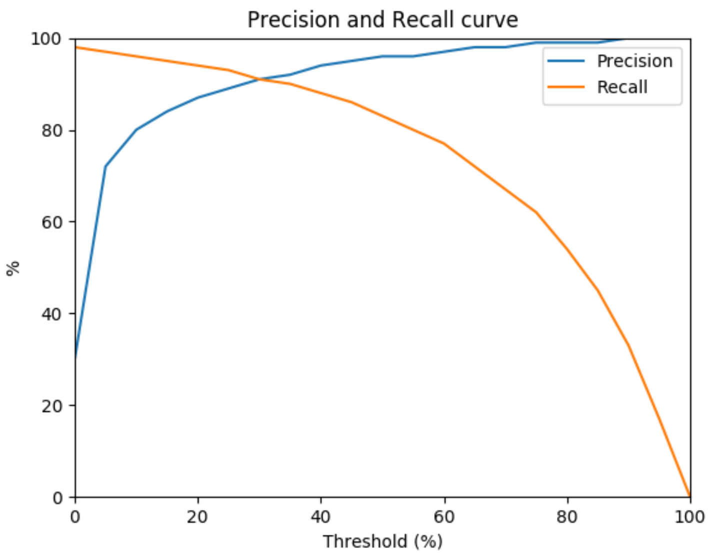


Figure 6.2: Precision and Recall curves for Tiny-YOLO-V3.

learned detection CNN, with precision quickly rising with the increase of the threshold, while recall starts near 100%, steadily dropping as the threshold is increased.

The Precision-Recall curve is depicted in Figure 6.3. In this figure it is possible to observe that the trained model nearly behaves like a perfect detector, with precision keeping a value close to 100% for almost all values of recall, dropping only when recall is itself approaching 100%. From this figure it is easier to understand the 95.54% mAP value, as the area under the curve covers almost the entire graph.

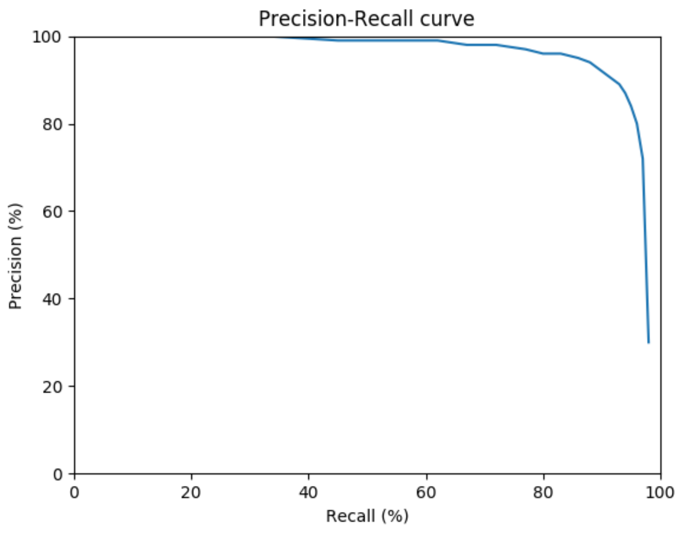


Figure 6.3: Precision-Recall curve for Tiny-YOLO-V3.

By observing some detection examples on test set images it is possible to confirm the efficiency of the network suggested by this data. In these images, presented in Figure 6.4, many things can be observed: first how the network is able to detect so many zooplankton beings of such different scales on the same image, even on noisy images generated during data augmentation, as the one in the bottom left corner. It is also possible to observe that the network is able to detect even transparent examples, such as gelatinous zooplankton species, as shown by the image at the bottom right corner. It is confirmed that the network didn't just learn to detect non-background objects, as can be seen in the middle right panel, where the image contains artifacts that the network didn't classify as zooplankton, indicating that it learned meaningful features of the class.

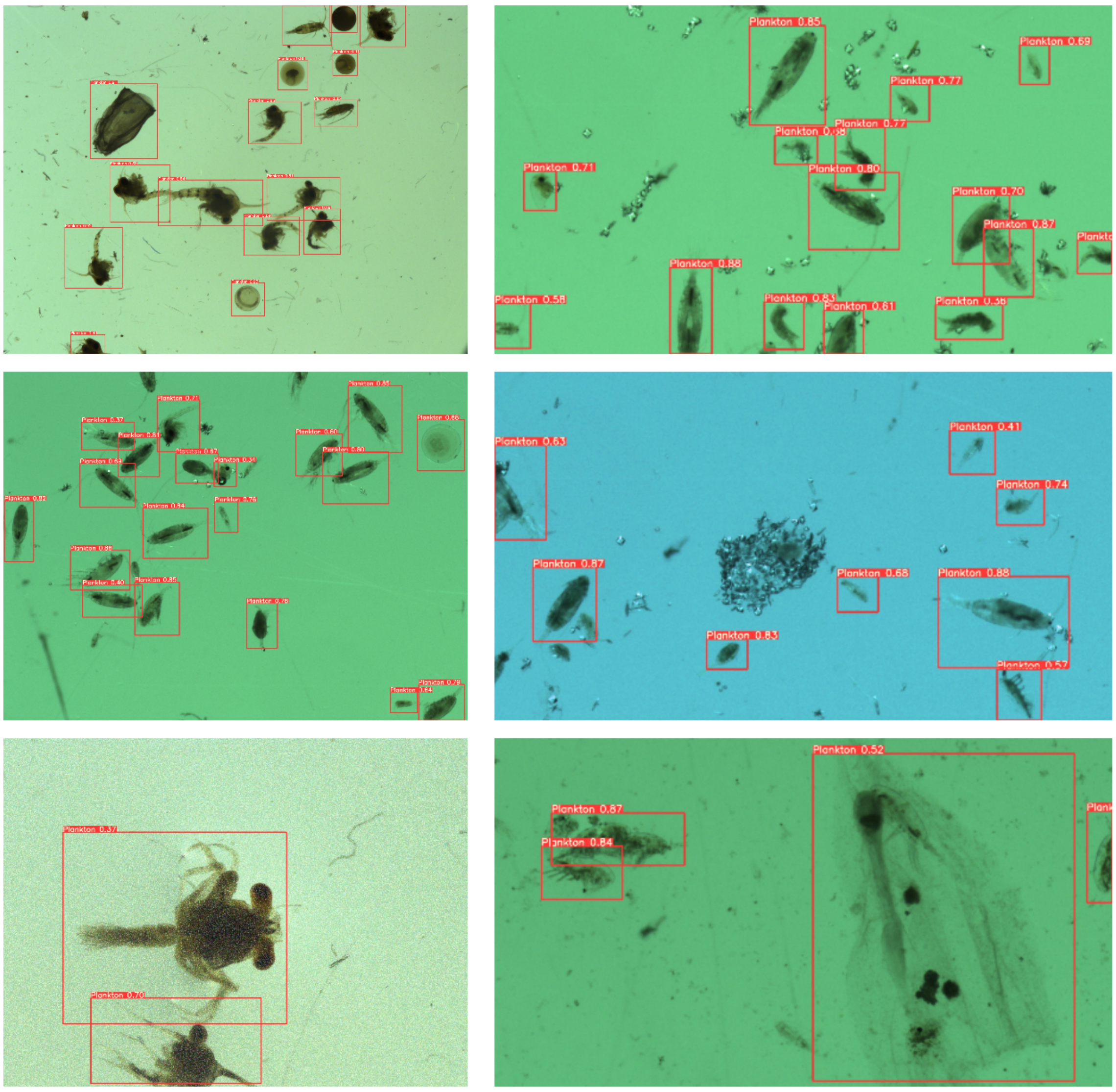


Figure 6.4: Detection results of Tiny-YOLO-V3.

6.2 Classification

In order to evaluate the proposed training for the classification networks provided in Table 5.3 an analysis on the losses for each network over iterations shown in Figure 6.5 can be done. From the analysis of this figure alone a few interesting remarks can be made. The first is that almost all networks seem to have stabilized both train and validation errors for the given training time, perhaps with the exception of VGG-16, which seems that could have benefited from more training time, despite being the network that by far

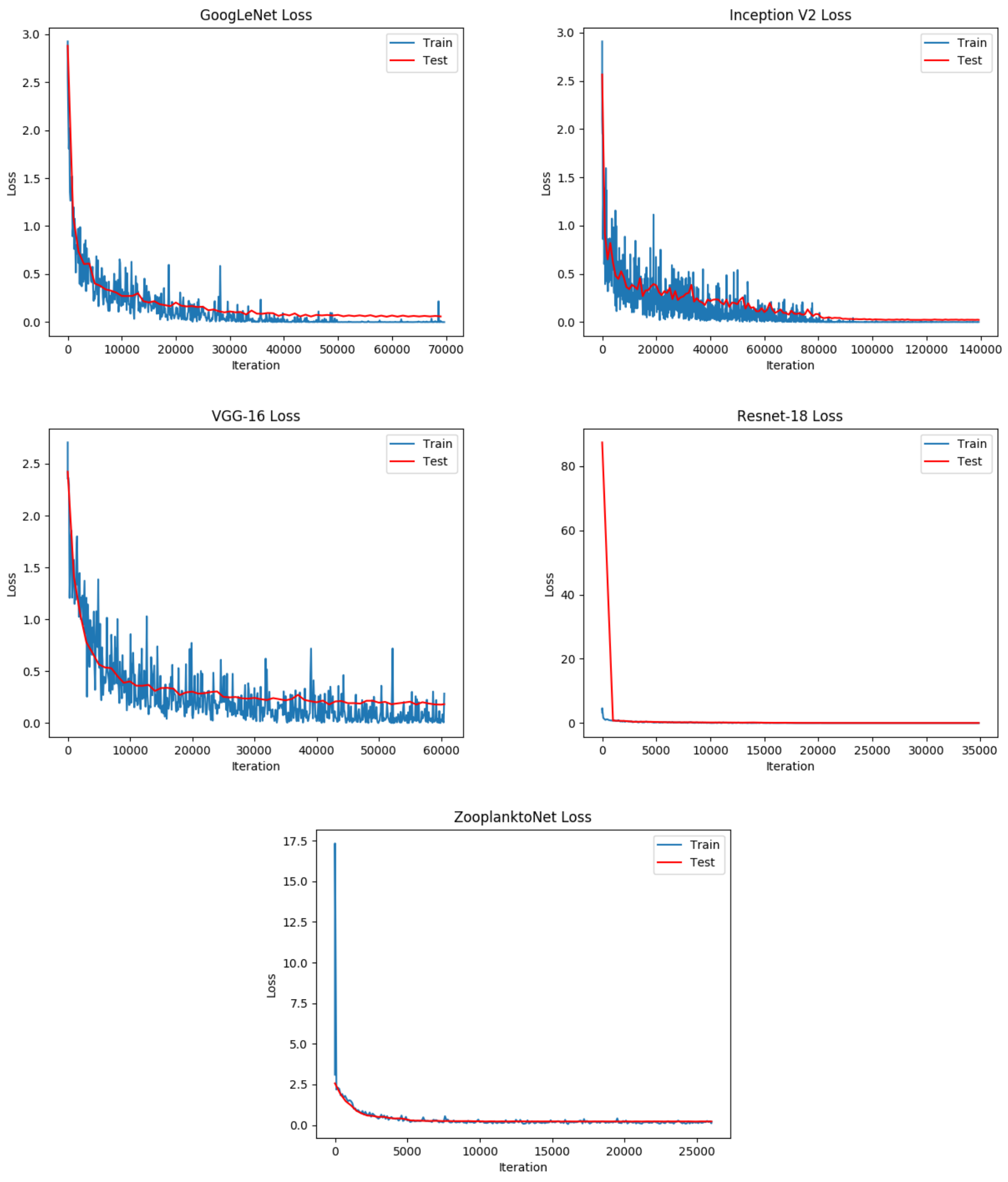


Figure 6.5: Classification CNNs training and test loss during training.

took longer to train, by far less epochs. This comes to confirm what was mentioned in the fundamentals that VGG-16 is one of the most computationally expensive CNNs. Another interesting remark is how fast both ZooplanktoNet and especially Resnet-18 seem to have reached loss stability. This can be explained by the fact that these two networks are the least complex of the lot, and by the batch size implemented for these netwoks, that was considerably larger than the ones used for the remaining CNNs. Batch size is also the reason why the loss decay is much smoother in these two networks.

In Figure 6.6 it is possible to observe the top-1 and top-5 accuracies over iterations these networks achieved for the ZooScan dataset.

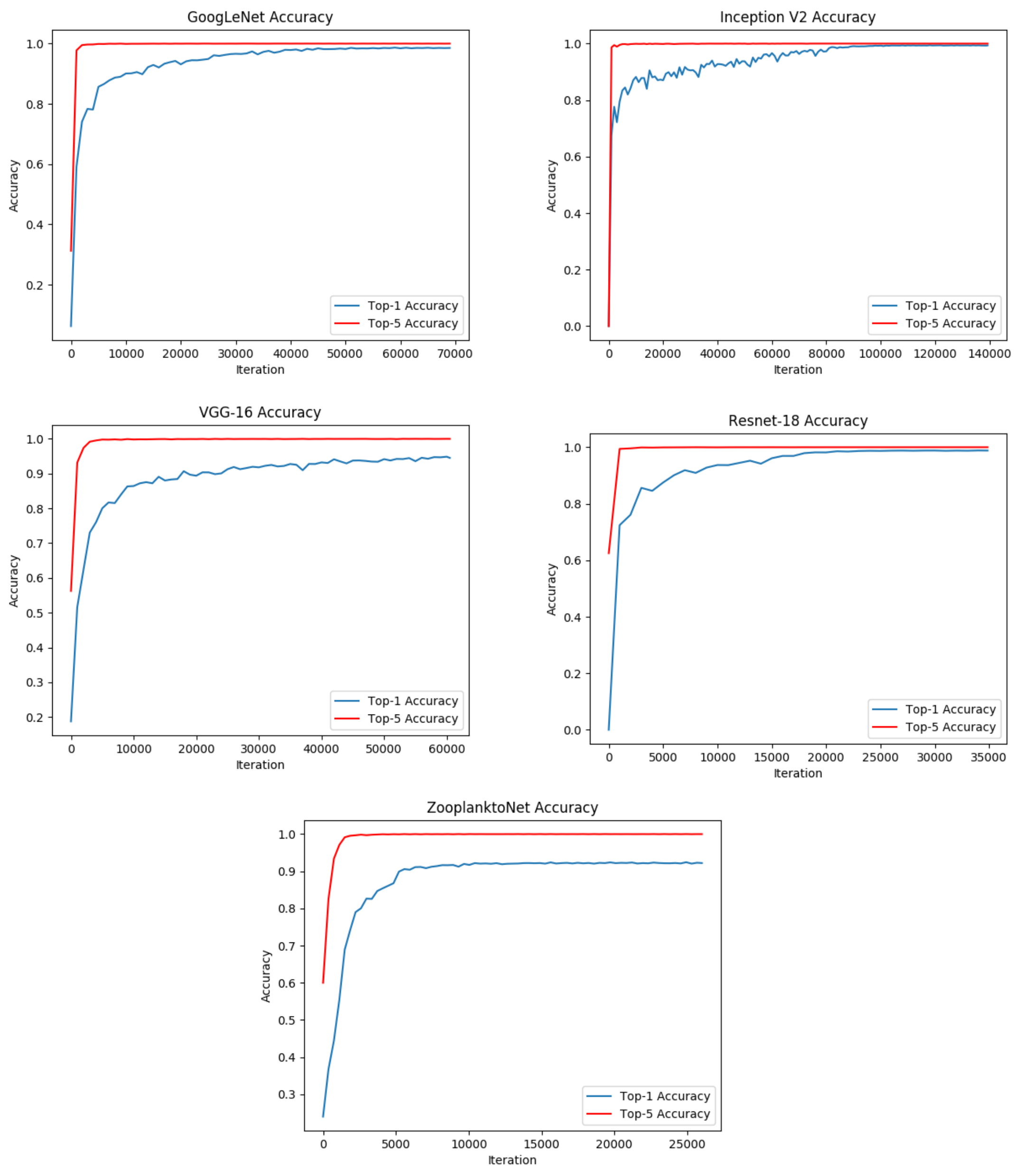


Figure 6.6: Classification networks top-1 and top-5 accuracies over iteration.

These graphs confirm some of the observations made about the loss graphs. It reinforces the idea that VGG-16 was still increasing the top-1 accuracy when training stopped. From these it is also visible that ZooplanktoNet seems to fall behind every other network in the top-1 accuracy, despite having its accuracy values stabilized, but reached the final value first. As for the top-5 accuracies all networks seem to have excelled, but it is not surprising, given the small number of classes in the dataset. With 13 classes it would be unlikely that a trained network didn't predict the real classes to be in the top 5 predictions.

The final loss and accuracy values can be seen in Table 6.1, with Inception-V2 being the best performing network in overall accuracy, and the only one that passed the 99% mark, but with all networks achieving impressive values. VGG-16 was the second worst performer of the lot, but as previously seen it hadn't finished training properly.

Table 6.1: Training results

Network	Train loss	Test loss	top-1 Accuracy	top-5 Accuracy
GoogLeNet	**1.90e-05**	0.059	98.53%	99.99%
Inception-V2	0.0012	**0.023**	**99.37%**	99.99%
Resnet-18	0.00047	0.037	98.82%	**100%**
VGG-16	0.2847	0.1828	94.51%	99.99%
ZooplanktoNet	0.1953	0.2219	92.19%	99.97%

In Table 6.2 the superiority of Inception-V2 is further evidenced by showing that it achieved the highest per class accuracy for all but one class.

Table 6.2: Per class accuracy

Class	GoogLeNet	Inception-V2	Resnet-18	VGG-16	ZooplanktoNet
Appendicularia	98.93%	**99.51%**	99.03%	94.56%	95.63%
Bubble	**100%**	**100%**	**100%**	**100%**	98.78%
Chaetognatha	98.92%	**99.54%**	99.39%	97.70%	96.31%
Cladocera Penilia	99.62%	**99.89%**	**99.89%**	99.68%	99.24%
Copepoda	99.05%	99.59%	**99.63%**	96.17%	95.28%
Decapoda	99.19%	**99.64%**	99.46%	96.60%	95.00%
Doliolida	98.95%	**99.82%**	**99.82%**	98.60%	97.02%
Egg	98.96%	**100%**	99.70%	99.11%	98.52%
Fiber	97.04%	**99.01%**	98.52%	92.11%	88.16%
Gelatinous	98.79%	**99.48%**	99.05%	96.45%	92.12%
Multiple	95.59%	**97.17%**	93.00%	73.25%	60.93%
Nonbio	98.30%	**99.32%**	98.79%	95.00%	92.57%
Pteropoda	98.06%	**99.76%**	**99.76%**	88.83%	91.75%

The per class performance of all networks is further evidenced in Figure 6.7, where the confusion matrices normalized per class of all networks are shown, which displays in a more easily manner the number of correct predictions per class, compared with the true label.

As for the networks performance on the datasets of extracted images from the detection network, both without preprocessing and with different values of adaptive thresholding, their accuracies are found in Table 6.3.

Table 6.3: Accuracy on Detection extracted images with different threshold values

	Unprocessed	Threshold 100	Threshold 128	Threshold 170	Threshold 200
GoogLeNet	7.39%	18.5%	33.9%	**41.5%**	11.0%
Inception-V2	21.1%	0%	2.34%	**27.6%**	16.2%
Resnet-18	6.2%	3%	5.8%	**15.1%**	7.3%
VGG-16	45.7%	50.5%	**66.1%**	65.4%	40.4%
ZooplanktoNet	15.0%	29.6%	**56.7%**	51.7%	15%

From the analysis of this table it is possible to see that overall performance becomes poor on these new images, which is not completely unexpected, given the networks were trained with a dataset that doesn't have much variation on their images. It is however evident that the pre-processing applied to the ROIs is effective in increasing the classification accuracy, with the values 128 and 170 spliting the best performances.

From Table 6.3 it is also interesting to observe that VGG-16 outperformed by far all other networks in all these new datasets, passing inclusively the 66% mark on the 128 value dataset and coming close to the 50% mark in all others. This seems to indicate that despite having one of the lowest accuracy values on the dataset it was trained with, it was actually the network that picked the most class relevant features, showing that it can generalize better to new data. This confirms what was told in Chapter 3 that VGG-16 is a great feature extractor, and explains why it is widely used for this task, despite being more computationally expensive.

The superiority of VGG-16 on the datasets created from the extracted ROIs from the detection network images over the remaining networks can be better visualized in Figure 6.8, where the confusion matrices for all networks are shown for the 170 threshold value dataset, which seems to be the one that got the best performances overall, despite not being the one that got the best result for VGG-16. In the VGG-16 related confusion matrix the values on the diagonal are much more evidenced, with 4 out of the 5 classes in the dataset getting more correctly predicted images than incorrect predictions, with only the "gelatinous" class failing to gather the most correct predictions. Note that this dataset only contained 5 classes, so the lack of more values along the diagonal is expected.

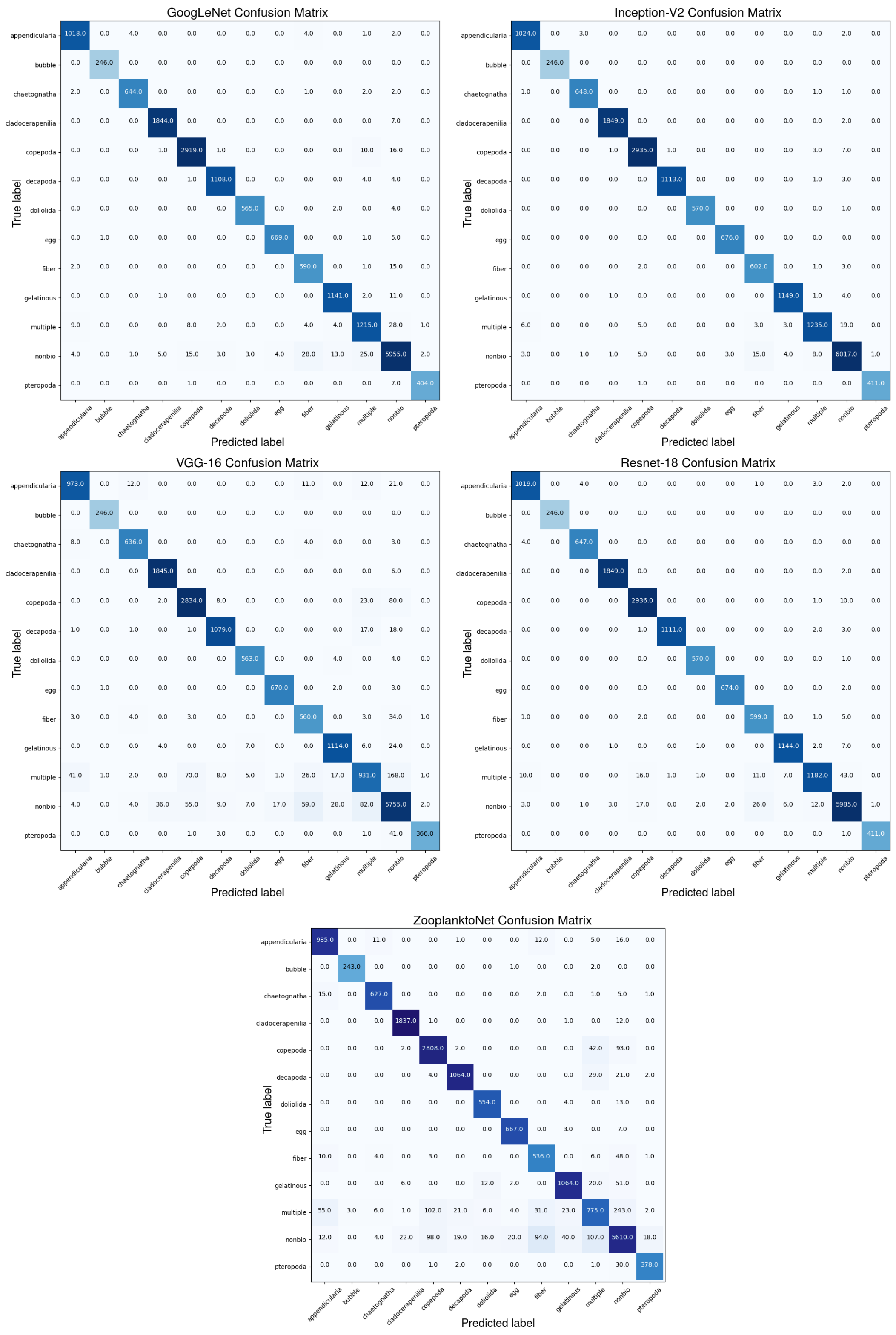
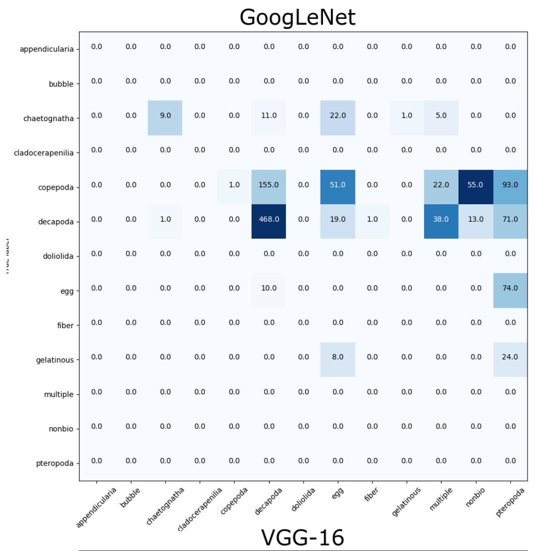
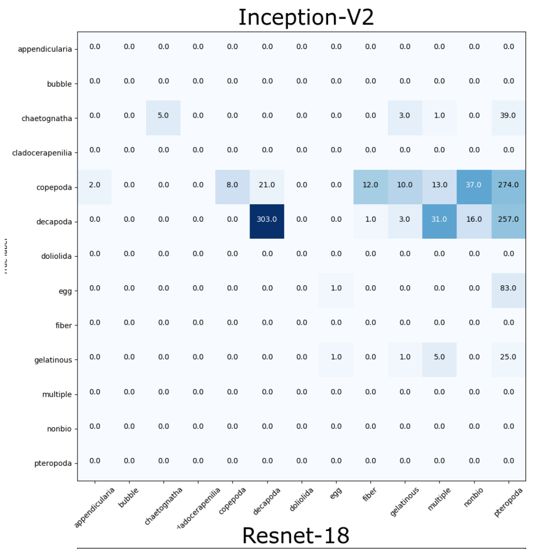
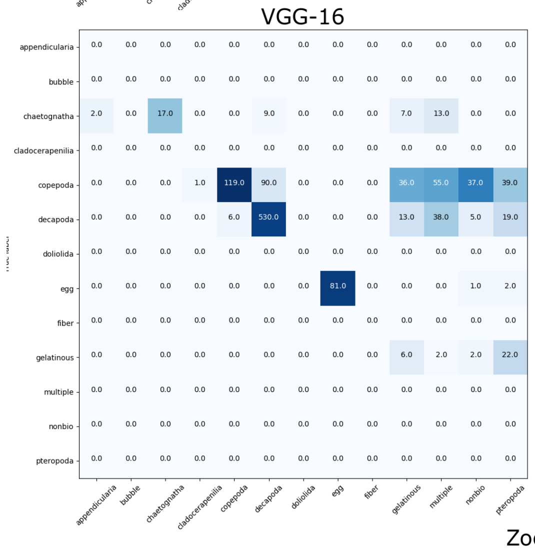
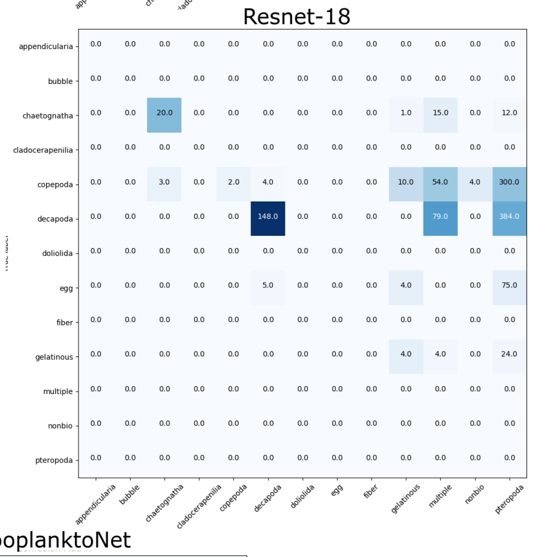


Figure 6.7: Confusion matrices of all classification networks.









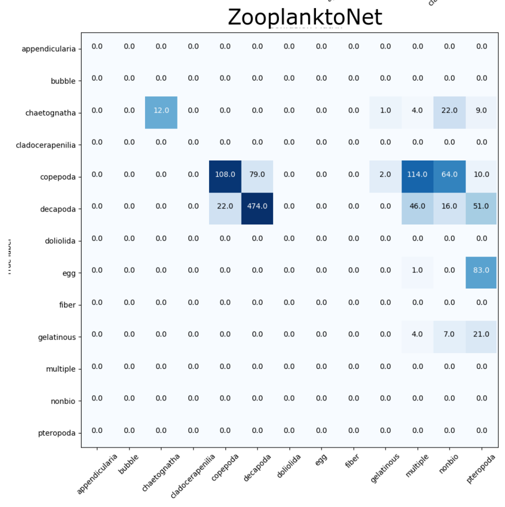


Figure 6.8: Confusion matrices of all classification networks on the 170 threshold dataset.

The last evaluation metric on the classification CNNs is the average forward pass time, which provides an insight on the time it takes for each CNN to output the classification results from an input image. These values are shown in Table 6.4. This test was run in one Nvidia RTX GeForce 2080 ti from the training workstation. From this it is visible that the Inception-V2 architecture obtained the best performance, closely followed by ZooplanktoNet.

Table 6.4: Average forward pass per Network

	GoogLeNet	Inception-V2	Resnet-18	VGG-16	ZooplanktoNet
Avg Pass time	34.66 ms	**22.30 ms**	48.94 ms	68.55 ms	24.04 ms

Chapter 7

Conclusions

Concluded the presentation of the results obtained from the proposed experiments, it is now time to do an overall evaluation on the work presented on this document, and how it covered the objectives proposed for the project.

With these experiments several different project objectives could be validated. System portability was guaranteed with the use of Movidius™NCS, given that once the software is installed in the machine, only the small NCS devices must be connected in the USB ports. This devices also allowed to perform real-time detection at a maximum of 7 FPS using MarinEye's imaging system. Portability was further improved by developing the detection and classification software as a ROS node that subscribes to camera images and publishes inference results.

Testing of the system *in situ* must be done in the future, however, with the data augmentation process done, images aren't expected to differ much from the ones captured with the experimental setup, being the water turbidity and the sparsity of the beings the main difference, while the latter actually helps improve frame rate.

Despite being trained with a dataset different from the one obtained with MarinEye imaging system, it was shown that with minor processing of the obtained ROIs it is possible to slightly increase accuracy. Nonetheless, the classification results can largely be improved when trained on a dataset obtained from the detection network, and a slight increase of frame rate is expected since minor pre-processing must be implemented. If such a dataset is obtained, the classification network can even be discarded if the detection CNN is also able to provide good classification accuracy in a multi-class labeled dataset.

The use of Tiny-YOLO-V2 shown that it's possible to perform zooplankton detection with CNNs with great accuracy using laboratory samples. By using CNNs for zooplankton detection MarinEye innovated in the *in situ* plankton imaging systems, being the first to implement CNNs for this purpose, as far as the research done allowed to assess.

7.1 Achievements and Future Work

Some of the work developed in this dissertation resulted in a publication of a conference paper for the MTS/OES OCEANS 2019 conference held in Marseille, France. The submitted abstract got accepted for the conference's Student Poster Competition, being one of the 18 chosen projects out of 59 applicants from 16 different countries. In this competition the accepted candidates had to create a poster to present the developed work to the conference attendants, as well as to present it to a panel of juries that would select the three best papers for a monetary prize. The resulting paper can be visualized in [58].

For future work, it will be necessary to expand the datasets. The creation of a labelled detection dataset with more classes than "zooplankton" would be a good improvement on the project, which would allow the system to perform both detection and classification with a single network, leading to an increase in performance, at least in terms of frame rate. It would also be interesting to test the system "as is" *in situ*, to evaluate the robustness of the work developed in a real world experiment.

References

[1] Christian Sardet. *Plankton Chronicles.* URL: http://planktonchronicles.org/en/ (visited on Nov. 12, 2020).

[2] NOAA. *How much oxygen comes from the ocean?* URL: https://oceanservice.noaa.gov/facts/ocean-oxygen.html (visited on Oct. 14, 2020).

[3] NASA Earth Observatory. *What are Phytoplankton?* URL: https://earthobservatory.nasa.gov/features/Phytoplankton (visited on Oct. 14, 2020).

[4] David Rissik and Iain Suthers. *Plankton: A Guide to Their Ecology and Monitoring for Water Quality.* June 2009. ISBN: 9780643097131. DOI: 10.1071/9780643097131.

[5] George Hendrey. "Acid Rain and Deposition". In: Dec. 2001, pp. 1–15. ISBN: 9780122268656. DOI: 10.1016/B0-12-226865-2/00001-8.

[6] A. Martins et al. "MarinEye: A tool for marine monitoring". In: *OCEANS 2016 - Shanghai.* Apr. 2016, pp. 1–7. DOI: 10.1109/OCEANSAP.2016.7485624.

[7] Moritz Sebastian Schmid et al. "The LOKI underwater imaging system and an automatic identification model for the detection of zooplankton taxa in the Arctic Ocean". In: *Methods in Oceanography* 15-16 (2016). Computer Vision in Oceanography, pp. 129–160. ISSN: 2211-1220. DOI: https://doi.org/10.1016/j.mio.2016.03.003. URL: http://www.sciencedirect.com/science/article/pii/S2211122015300050.

[8] Jan Schulz et al. "Imaging of plankton specimens with the lightframe on-sight keyspecies investigation (LOKI) system". In: *Journal of the European Optical Society Rapid Publications* 5 (Apr. 2010), 10017s. DOI: 10.2971/jeos.2010.10017s.

[9] Moritz Schmid et al. "ZOOMIE v 1.0 (Zooplankton Multiple Image Exclusion)". In: (May 2015). DOI: 10.5281/zenodo.17928.

[10] Leo Breiman. "Random Forests". In: *Machine Learning* 45 (2001). DOI: 10.1023/A:1010933404324.

[11] Cabell Davis et al. "A three-axis fast-tow digital Video Plankton Recorder for rapid surveys of plankton taxa and hydrography". In: *Limnology and Oceanography: Methods* 3 (Feb. 2005). DOI: 10.4319/lom.2005.3.59.

[12] C. Davis et al. "Real-time observation of taxa-specific plankton distributions: An optical sampling method". In: *Marine Ecology-progress Series - MAR ECOL-PROGR SER* 284 (Dec. 2004), pp. 77–96. DOI: 10.3354/meps284077.

[13] Teuvo Kohonen. "Learning Vector Quantization". In: *Self-Organizing Maps*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 175–189. ISBN: 978-3-642-97610-0. DOI: 10.1007/978-3-642-97610-0_6. URL: https://doi.org/10.1007/978-3-642-97610-0_6.

[14] Hongsheng Bi et al. "Deployment of an imaging system to investigate fine-scale spatial distribution of early life stages of the ctenophore Mnemiopsis leidyi in Chesapeake Bay". In: *Journal of Plankton Research* 35.2 (Dec. 2012), pp. 270–280. ISSN: 0142-7873. DOI: 10.1093/plankt/fbs094. eprint: https://academic.oup.com/plankt/article-pdf/35/2/270/11495679/fbs094.pdf. URL: https://doi.org/10.1093/plankt/fbs094.

[15] Hongsheng Bi et al. "A Semi-Automated Image Analysis Procedure for In Situ Plankton Imaging Systems". In: *PLOS ONE* 10.5 (May 2015), pp. 1–17. DOI: 10.1371/journal.pone.0127121.

[16] Lorenzo Corgnati et al. "Looking inside the Ocean: Toward an Autonomous Imaging System for Monitoring Gelatinous Zooplankton". In: *Sensors* 16.12 (Dec. 2016), p. 2124. ISSN: 1424-8220. DOI: 10.3390/s16122124. URL: http://dx.doi.org/10.3390/s16122124.

[17] Simone Marini et al. "GUARD1: An autonomous system for gelatinous zooplankton image-based recognition". In: May 2015. DOI: 10.1109/OCEANS-Genova.2015.7271704.

[18] CHDK. *Canon Hack Development Kit*. URL: https://chdk.fandom.com/wiki/CHDK (visited on Nov. 6, 2020).

[19] *Raspberry Pi*. URL: https://www.raspberrypi.org/ (visited on Nov. 6, 2020).

[20] Ali Reza. "Realization of the Contrast Limited Adaptive Histogram Equalization (CLAHE) for Real-Time Image Enhancement". In: *VLSI Signal Processing* 38 (Aug. 2004), pp. 35–44. DOI: 10.1023/B:VLSI.0000028532.53893.82.

[21] D. Walther, D. R. Edgington, and C. Koch. "Detection and tracking of objects in underwater video". In: *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.* Vol. 1. 2004, pp. I–I. DOI: 10.1109/CVPR.2004.1315079.

[22] Christine Mol et al. "A Regularized Method for Selecting Nested Groups of Relevant Genes from Microarray Data". In: *Journal of computational biology : a journal of computational molecular cell biology* 16 (June 2009), pp. 677–90. DOI: 10.1089/cmb.2008.0171.

[23] J.R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press:Cambridge, MA, USA, 1992.

[24] Lorenzo Corgnati et al. "Automated Gelatinous Zooplankton Acquisition and Recognition". In: Aug. 2014. DOI: 10.1109/CVAUI.2014.12.

[25] Eric C. Orenstein et al. "The Scripps Plankton Camera system: A framework and platform for in situ microscopy". In: *Limnology and Oceanography Methods* (2020). DOI: 10.1002/lom3.10394. URL: https://app.dimensions.ai/details/publication/pub.1131430628%20and%20https://aslopubs.onlinelibrary.wiley.com/doi/pdfdirect/10.1002/lom3.10394.

[26] Philippe Grosjean et al. "Enumeration, measurement, and identification of net zooplankton samples using the ZOOSCAN digital imaging system". In: 61 (June 2004), pp. 518–525.

[27] Gaby Gorsky et al. "Digital zooplankton image analysis using the ZooScan integrated system". In: *Journal of Plankton Research* 32.3 (Mar. 2010), pp. 285–303. ISSN: 0142-7873. DOI: 10.1093/plankt/fbp124. eprint: http://oup.prod.sis.lan/plankt/article-pdf/32/3/285/4394627/fbp124.pdf. URL: https://doi.org/10.1093/plankt/fbp124.

[28] J. Dai et al. "ZooplanktoNet: Deep convolutional network for zooplankton classification". In: *OCEANS 2016 - Shanghai.* Apr. 2016, pp. 1–6. DOI: 10.1109/OCEANSAP.2016.7485680.

[29] Jialun Dai et al. "A Hybrid Convolutional Neural Network for Plankton Classification". In: (Mar. 2017), pp. 102–114.

[30] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems.* Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012.

[31] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *International Conference on Learning Representations.* 2015.

[32] J. Cui et al. "Texture and Shape Information Fusion of Convolutional Neural Network for Plankton Image Classification". In: *2018 OCEANS - MTS/IEEE Kobe Techno-Oceans (OTO).* May 2018, pp. 1–5. DOI: 10.1109/OCEANSKOBE.2018.8559156.

[33] Eric C. Orenstein et al. "WHOI-Plankton- A Large Scale Fine Grained Visual Recognition Benchmark Dataset for Plankton Classification". In: *CoRR* abs/1510.00745 (2015). arXiv: 1510.00745. URL: http://arxiv.org/abs/1510.00745.

[34] Pedro Domingos. *The Master Algorithm: How the Quest for the Ultimate Learning Machine Will Remake Our World*. USA: Basic Books, Inc., 2018. ISBN: 0465094279.

[35] Thomas M. Mitchell. *Machine Learning*. 1st ed. USA: McGraw-Hill, Inc., 1997. ISBN: 0070428077.

[36] F. Rosenblatt. *The Perceptron - a perceiving and recognizing automaton*. Report 85-460-1. Cornell Aeronautical Laboratory, 1957.

[37] Jagreet Kaur Gill. *Log Analytics Tools and Automating with Deep learning - XenonStack*. URL: `https://www.xenonstack.com/blog/log-analytics-deep-machine-learning/` (visited on Nov. 19, 2020).

[38] Donald O. Hebb. *The organization of behavior: A neuropsychological theory*. Wiley, 1949.

[39] Matthew D. Zeiler. *ADADELTA: An Adaptive Learning Rate Method*. 2012. arXiv: `1212.5701 [cs.LG]`.

[40] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *Journal of Machine Learning Research* 12 (July 2011), pp. 2121–2159.

[41] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: `1412.6980 [cs.LG]`.

[42] MathWorks. *What is a Convolutional Neural Network?* URL: `https://www.mathworks.com/discovery/convolutional-neural-network-matlab.html` (visited on Oct. 21, 2021).

[43] Matthew D Zeiler and Rob Fergus. *Visualizing and Understanding Convolutional Networks*. 2013. arXiv: `1311.2901 [cs.CV]`.

[44] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: `10.1109/5.726791`.

[45] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: `10.1007/s11263-015-0816-y`.

[46] Christian Szegedy et al. *Going Deeper with Convolutions*. 2014. arXiv: `1409.4842 [cs.CV]`.

[47] Neurohive. *VGG16 – Convolutional Network for Classification and Detection*. URL: `https://neurohive.io/en/popular-networks/vgg16/` (visited on Oct. 27, 2021).

[48] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: `1512.03385 [cs.CV]`.

[49] Andrew G. Howard et al. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. 2017. arXiv: `1704.04861 [cs.CV]`.

[50] Prakhar Ganesh. *Types of Convolution Kernels : Simplified.* URL: https://towardsdatascience.com/types-of-convolution-kernels-simplified-f040cb307c37 (visited on Oct. 28, 2021).

[51] Joseph Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection.* 2016. arXiv: 1506.02640 [cs.CV].

[52] Joseph Redmon and Ali Farhadi. *YOLO9000: Better, Faster, Stronger.* 2016. arXiv: 1612.08242 [cs.CV].

[53] Joseph Redmon and Ali Farhadi. *YOLOv3: An Incremental Improvement.* 2018. arXiv: 1804.02767 [cs.CV].

[54] Wei Liu et al. "SSD: Single Shot MultiBox Detector". In: *Lecture Notes in Computer Science* (2016), pp. 21–37. ISSN: 1611-3349. DOI: 10.1007/978-3-319-46448-0_2. URL: http://dx.doi.org/10.1007/978-3-319-46448-0_2.

[55] João Resende et al. "Autonomous High-Resolution Image Acquisition System for Plankton". In: *2021 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC).* 2021, pp. 74–78. DOI: 10.1109/ICARSC52212.2021.9429789.

[56] Robert K Cowen et al. "Planktonset 1.0: Plankton imagery data collected from fg walton smith in straits of florida from 2014–06-03 to 2014–06-06 and used in the 2015 national data science bowl (ncei accession 0127422)". In: *NOAA National Centers for Environmental Information* (2015).

[57] Robert K. Cowen and Cédric Guigand. "In situ Ichthyoplankton Imaging System(ISIIS): system design and preliminary results". In: *Limnology and Oceanography: Methods* 6 (Feb. 2008). DOI: 10.4319/lom.2008.6.126.

[58] Pedro Geraldes et al. "In situ real-time Zooplankton Detection and Classification". In: *OCEANS 2019 - Marseille.* 2019, pp. 1–6. DOI: 10.1109/OCEANSE.2019.8867552.