TÉCNICO LISBOA

AI-based Visual Navigation Solution for Autonomous Underwater Vehicles

Tiago Martim Gomes Alves

Thesis to obtain the Master of Science Degree in

Aerospace Engineering

Supervisors: Prof. Rodrigo Martins de Matos Ventura
Dr. Nuno Tiago Salavessa Cardoso Hormigo Vicente

Examination Committee

Chairperson: Prof. Paulo Jorge Coelho Ramalho Oliveira
Supervisor: Prof. Rodrigo Martins de Matos Ventura
Member of the Committee: Prof. José Alberto Rosado dos Santos Victor

October 2021

À memória de Fernando Correia, pelas longas conversas e pela sabedoria transmitida.

Acknowledgments

I would like to begin by thanking my supervisors, Professor Rodrigo Ventura and Tiago Hormigo from Spin.Works. Computer vision and deep learning were unexplored fields for me when I started developing this thesis. Their guidance and insights were invaluable in formulating the research questions and methodology. João Oliveira, from Spin.Works, was also crucial on enlightening me about photogrammetry and computer vision basics.

I also have to acknowledge the ROV Luso team, from EMEPC, for providing the video footage necessary to create the dataset developed in this work.

I want to thank my family. You have always been available to listen and encourage me. Last but not least, I want to give a warm thank you to the friends I made along this journey at Técnico, specially Diogo, Francisco, João Ferreira, João Moita, Pedro and Rita.

Resumo

Veículos subaquáticos autónomos têm a capacidade de mudar a forma como exploramos as zonas mais profundas do oceano e, possivelmente, o espaço. Atualmente uma parte significativa da pesquisa oceanográfica é feita com veículos operados remotamente, que são lançados de um navio ou de um ponto na costa, estando ligados a uma interface, a partir de onde um piloto o pode conduzir pelo ambiente a explorar. Esta abordagem apresenta algumas preocupações quando utilizar um cabo é pouco prático. Por exemplo, na exploração de áreas embaixo de calotes polares, missões de defesa e vigilância, entrega de carregamentos ou análise de catástrofes ambientais (petróleo, gás natural). Em relação ao espaço, um dos grandes interesses de pesquisa das agências internacionais foca-se em encontrar ambientes capazes de suportar vida no nosso sistema solar, como pode ser o caso de luas geladas - Enceladus, Europa ou Titan.

Este trabalho foca-se em testar o impacto de introduzir uma abordagem com fundamentos de Inteligência Artificial em navegação visual para veículos totalmente autónomos. Para atingir estes objetivos, há vários obstáculos a ultrapassar. Primeiro, é essencial ter um dataset com pares de imagens e máscaras de segmentação, para treinar uma rede neuronal adequada a segmentação. Segundo, escolher uma rede neuronal adequada à tarefa de segmentar imagens, treiná-la e testá-la com diferentes parâmetros para determinar a configuração que nos trará melhores resultados. Por último, avaliar se estes métodos melhoram a performance the algoritmos de navegação visual.

Para criar a base de dados, precisamos de imagens e de uma ferramenta que auxilie o processo de criação das máscaras *ground truth*. Esta ferramenta deve fazer uma boa estimativa de uma máscara de segmentação, que será posteriormente corrigida à mão para obter resultados mais precisos.

O modelo de deep learning escolhido para a tarefa de segmentação foi uma *Fully Convolutional Network* (FCN), que é um robusto modelo de ponta. Com esta escolha foi possível atingir valores de *pixel accuracy* que rondam os 93% e *Intersections over Union* a rondar 85%.

Esta solução foi testada em algoritmos de SLAM online e offline, ao sobrepor as imagens originais com as máscaras de segmentação. Em relação aos testes com SLAM offline, foi utilizado um software bsatante robusto, pelo que não foi possível constatar uma diferença significativa em termos de métricas quantitativas, comparando os casos em que foram utilizadas máscaras com os casos em que não se utilizou. No entanto, numa análise mais qualitativa, a reconstrução da zona foi mais precisa com máscaras, se compararmos com as imagens originais. Por fim, testou-se num algorítmo de navegação visual em tempo real, o ORB-SLAM. Para este caso, os resultados foram melhores utilizando as máscaras, quer em termos quantitativos como qualitativos.

Palavras-chave: Veículos Subaquáticos Autónomos, Deep Learning, Segmentação, SLAM

Abstract

Autonomous underwater vehicles have the potential to be a game changer in deep ocean and space exploration. In present days most of ocean exploration is done with remotely operated vehicles (ROV's), deployed from a vessel and connected through cables to an interface where a pilot can operate the vehicle. This poses several challenges when a cable-based solution is not ideal, like beneath ice sheets, defense and surveillance missions, payload deliveries or assessment of environmental hazards. As for space exploration, one of the great interests of today's research is finding environments capable of bearing life within our solar systems, as is possibly the case of ocean worlds - Enceladus, Europa or Titan.

This research aims at testing the impact of introducing an AI-based approach for visual navigation. To achieve this several challenges have to be overtaken. First, an annotated dataset with pairs of input images and segmentation grounds truths is essential for training an artificial neural network suited for semantic segmentation. Second, choosing a state of the art neural network for segmentation, then training and testing it with different parameters to determine what configuration provides the best results. Finally, evaluate if this methodology improves the accuracy of visual navigation and scene reconstruction algorithms, such as online and offline SLAM.

To create the dataset, there is the need for a tool that outputs a good initial prediction of a segmentation mask, which should subsequently hand corrected for increased accuracy.

The deep learning model used for semantic segmentation was a Fully Convolutional Netwok (FCN), which is a robust state of the art model, that takes a pre-trained convolutional neural network and augments the pixel-wise predictions to a segmentation map. This model achieved an accuracy close to 93%.

This methodology was then tested on offline and online SLAM algorithms, by coupling the original frames with the respective segmentation masks yielded by the FCN. For the offline SLAM tests, a very robust software was used, therefore in terms of quantitative metrics there was no expressive difference when masks were used, but scene reconstruction was much more accurate on a qualitative analysis by comparing with the initial footage. This implementation was then tested on ORB-SLAM, an online SLAM algorithm, and the results were better when using masks, on qualitative and quantitative basis.

Keywords: Autonomous Underwater Vehicles, Deep Learning, Segmentation, SLAM

Contents

List of Tables

List of Figures

Nomenclature

Roman symbols

AI	Artificial Intelligence
AUV	Autonomous Underwater Vehicle
CNN	Convolutional Neural Network
DL	Deep Learning
FCN	Fully Convolutional Network
ILSRVC	ImageNet Large Scale Visual Recognition Challenge
IoU	Intersection over Union
RMSProp	Root Mean Squared Propagation
ROV	Remotely Operated Vehicle
SLAM	Simultaneous Localization and Mapping
UUV	Unmanned Underwater Vehicle

Chapter 1

Introduction

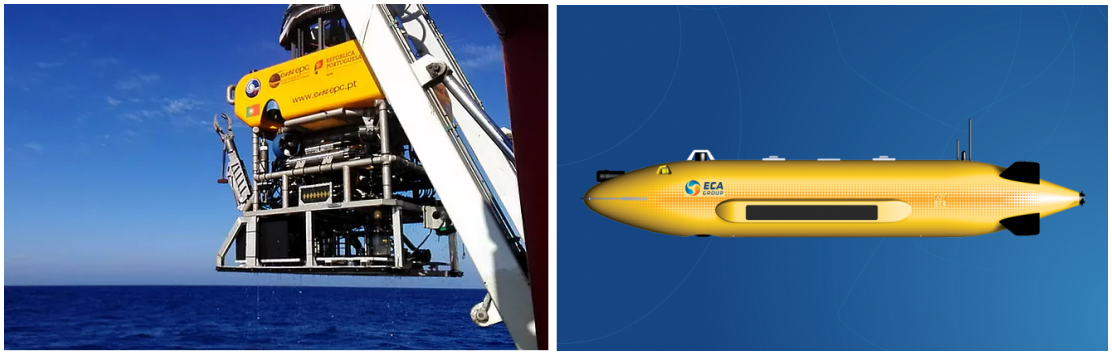
Contents

1.1 Motivation

Autonomous underwater vehicles (AUVs) can revolutionize deep sea exploration, by changing the way data is acquired for further mapping and monitoring. Nowadays, unmanned underwater vehicles (UUVs), are usually deployed from a vessel through cables connected to an interface, operated by a pilot responsible for guiding the vehicle across the explored environment, such is the case of remotely operated vehicles (ROVs). This approach is progressively being replaced by fully autonomous vehicles that can operate when a cable-based solution is not possible, for instance beneath ice sheets in polar regions. Vehicles capable of navigating autonomously are also more valuable for defense purposes, e.g. surveillance, reconnaissance or payload deliveries, assessment of environmental hazards related to oil and gas. Space exploration can also benefit from AUVs. One of the goals of several international organizations is to explore parts of our solar system potentially capable of hosting life, as is the case of ocean worlds, such as Enceladus, Europa or Titan.

Developing fully autonomous systems is crucial to achieving the above goals. Tim Shank, a researcher from WHOI, explains his vision in a conference [1], where a series of vehicles equipped with sensors are capable of communicating with each other and with local base stations about positioning, sensing and samples they are collecting. Russel Smith, an engineer from NASA's Jet Propulsion Lab

[1] https://www.youtube.com/watch?v=BGD_oyPGT6w&ab_channel=oceanexplorergov



(a) ROV LUSO. Source: EMEPC (b) Example of AUV. Source: ECA

Figure 1.1: Examples of UUVs.

(JPL), spoke in the same conference about the importance of vehicles capable of gathering visual information and use it for localization and mapping, enabling fully autonomous navigation. For space exploration to be successful in environments that are expected to be similar to the deep ocean areas we have on earth, developing robust technologies that are able to endure and prosper on test missions on our oceans is a clear first step to achieving this goal.

This thesis brings a visual navigation solution, powered by artificial intelligence algorithms. AI algorithms, as artificial neural networks, have rose in popularity in recent years, much due to contests like ImageNet Large Scale Visual Recognition Challenge (ILSVRC) where deep learning based solutions have been thriving and winning over solutions that are not AI-based. The purpose of integrating AI is to increase robustness and allow systems to correctly identify terrain, by segmentation, while coping with rapid changing environments, where lighting conditions are not optimal. If terrain is correctly identified and noise, bubbles and dust are discarded by segmentation, localization and mapping tools should be able to produce more accurate outcomes.

(a) (b)

Figure 1.2: Examples of the explored environment.

1.2 State of the art

The first documented autonomous underwater vehicle was developed as early as the 50s decade of the 20^{th} century, it was called Self-propelled Underwater Research Vehicle (SPURV) and it was capable of operating at depths between 3000 and 3600 meters, in missions lasting up to 4 hours. At that time, building these vehicles was very costly and the machines were big, heavy and inefficient, with the SPURV being able to travel around 2.5 m/s. Technology has evolved and today we can find sleeker high-end AUVs capable of travelling at 20 m/s, moving with 6 degrees of freedom, while being capable of monitoring and mapping the ocean floor [1].

One of the main challenges for AUVs is navigation and localization, since global position systems (GPS) are not usable, making position estimation a very difficult task. A study conducted by Stutters et al. [2] states that current navigation techniques for AUVs can be split into inertial, acoustic and geographical navigation.

Inertial navigation systems (INS) use sensors to estimate the vehicles speed and position. Magnetometers can be used to measure magnetic field and estime heading, accelerometers are used to measure acceleration and gyroscopes determine rotational data like velocity and acceleration. One disadvantage of inertial navigation systems is error accumulation. All the position estimates are determined by integration of data from the sensors. Sensor error can have a significant impact when introduced in these calculations, leading to loss of accuracy in position and velocity estimation. Therefore, these systems work better when paired with others, like depth sensor, compass, GPS or Sonar. Sonar is very expensive when compared with the others, but the AUV doesn't need to resurface when the GPS signal is lost, so it is better for missions where the AUV has to operate at great depths and the others are better for operations closer to surface.

The acoustic navigation makes use of time of flight concept and acoustic transponders to estimate the vehicle's position. Position is estimated relatively to a surface vessel or buoy equipped with an array of acoustic transponders and transducers. The signal's time of travel from the vessel to the AUV and back can be used to estimate distance, while phase shift of signals received by different transducers are useful to determine the vehicle's direction. Acoustic navigation also allows cooperative approaches, where a series of vehicles are equipped with INS and one is equipped with more sophisticated navigation technologies, such as SVL sonars, and acoustic transponders. The last is capable of estimating the relative position of the others, through acoustic signals, while transmitting more detailed navigation information acquired through its more advanced navigation technologies.

AUVs equipped with geographical navigation systems use references on the surroundings for guidance and localization and Simultaneous Localization and Mapping (SLAM) algorithms are widely used. Data can be acquired via optical tools, like monocular or multi-camera systems. Research conducted by Carrasco et al. [3] integrated a multi-camera system in an AUV equipped with other localization instruments and it showed improvements when compared to other configurations. A docking system for UAVs was also developed in 2019 by Liu et al. [4], where convolutional neural networks are used to detected possible docking sites. Another interesting work [5], published in 2019, presents a vision based

navigation solution for AUVs to follow pipelines, during monitoring and inspection tasks.

Another way to acquire geographical data is through sonar technologies, which have been the preferable tool for this type of navigation. Sonars are very useful in identifying different structures present in deep sea environments, like rocks, seafloor or mud. The outcome of a sonar is a bathymetry map, where hard structures, such as rocks, are presented in more intense or darker colors and softer structures are lighter. The main principle behind a sonar is reflection of sound waves, emitted by transponders in the AUV and refflected by seabed structures, that once capture by arrays of antennas on the vehicle can be used to determine distance and direction of obstacles [1].

In recent years, computer vision has been through an increase of popularity and applications, much due to the evolution of deep learning, a subfield of machine learning. Deep learning allows computers to perform tasks that for humans are very intuitive, yet difficult to hard-code in computer languages, such as understanding written and spoken language, or object identification and classification. [6] This surge of popularity and applications is powered by increasing computation capacity and the development of techniques like parallel computing, where graphics processing units (GPUs), which usually have more cores than CPUs, are used in the training of neural networks, a procedure that had serious limitations a few decades ago.

In the scope of computer vision is where deep learning has had its most expressive developments, more specifically with the introduction of convolutional neural networks (CNN). The winner of 2012's ILSVRC, the AlexNet [7], was the first deep learning powered algorithm to win the contest, setting a milestone for the years to come. The developers of this algorithm made some groundbreaking discoveries, such as reducing the time required in training by six times, through the usage of rectified linear unit (ReLU) activation function. One drawback of this architecture is the huge number of parameters and neurons, with almost $6,000,000$ of the first and $650,000$ of the last. In 2014 the VGG Net [8] research was published, with breakthroughs that save training time and the amount of network parameters, after winning the ILSVRC competition. The ResNet [9], short for Residual Network, came with new findings on network depth, stating that increasing the number of layers only improves model accuracy up to a certain level.

The evolution of convolutional neural networks for image classification tasks paved the way for AI-based solutions for semantic segmentation problems. Segmentation is in fact a classification problem, where each pixel is assigned to a certain class, therefore using classification methods for segmentation problem is becoming the state of the art practice [10]. R-CNN [11], a CNN based semantic segmentation algorithm, achieved great results on the PASCAL VOC dataset. Long et al. [12] started a Fully Convolutional Network (FCN) trend for segmentation tasks, with a model based on the VGG Net, which combines low layer and high layer results to improve accuracy. Other researches worth mentioning, which are very similar in architecture except on certain nuances that depend on the task they perform, are the U-Net [13], SegNet [14] and the DeconvNet [15].

1.3 Problem statement

Developing a visual based navigation solution for underwater vehicles operating in deep ocean comes with several challenges. Since the feature detection of this solution is based in deep learning, which requires annotated datasets for training, finding or developing a good dataset was the first task. The second challenge would be training a neural network suited for segmentation, in order to have a robust feature detection tool. At last, assess if the proposed methodology helps improve the performance of SLAM algorithms.

1.3.1 Overview of the approach

To tackle the dataset problem, a dataset with pairs of input images and labeled segmentation ground truth needs to be created, as there is a lack of deep sea annotated datasets publicly available. The footage will provided by ROV Luso from EMEPC (Task Group for the Extension of the Continental Shelf) in video format. After sampling the videos, a methodology for the ground truth creation will be be addressed, to accelerate the process of annotating the images, which would be extremely time consuming if done exclusively by hand.

To address the segmentation problem, we want to use state of the art technology, robust enough to endure the difficult navigation conditions of a deep underwater environment. One of the possibilities is to resort to neural networks, such as Fully Convolutional network.

To assess if the developed network contributes to SLAM algorithms, the outputs were tested, coupled with the input images, on both offline and online SLAM algorithms. Offline SLAM is used in post-exploration situations to reconstruct and map the visited sites. Online SLAM is applied for real-time autonomous navigation. Thus, evaluating on both tools is crucial in the scope of deep sea - and future ocean planets - exploration.

1.4 Contributions

The main contributions of this work are:

- Development of an annotated dataset of deep underwater environments rich in hydrothermal vents and sea bed footage;
- Training and testing a state of the art AI-based semantic segmentation algorithm with the created dataset;
- Evaluate the contribution of introducing AI in feature detection to improve the performance of navigation and reconstruction algorithms, such as SLAM.

1.5 Thesis Outline

This thesis focuses on applying deep learning algorithms for semantic segmentation and use this results to improve results of simultaneous localization and mapping algorithms. The workflow is described bellow.

The first chapter provides a short introduction to the problem being solved, an insight to the proposed methodology and a review on current state of the art methods.

The second chapter, the background, exposes important concepts about segmentation, artificial intelligence, deep learning and localization and mapping tools that are crucial for proper understanding of the following chapters.

In chapter 3, the reader can find the description of the implementation explaining how the problem was solved. From preprocessing of images to allow for correct dataset labeling, to the implementation of neural networks for semantic segmentation.

The results achieved by the proposed methodology are exposed on the fourth chapter, which exposes the performance of image segmentation neural network and the impact of these results in simultaneous localization and mapping tools.

The conclusions drawn from this work and ideas of future work can be found on chapter 5.

Chapter 2

Background

Contents

2.1 Semantic Segmentation

Image segmentation is a crucial computer vision task in order to better extract information from an image. It can be described as dividing an image in several parts, usually following specific criteria. Semantic segmentation is a task that consists of dividing an image into semantically relevant parts. In other words, split parts of an image into different classes (fig. 2.1)

Semantic segmentation is a very useful tool and is essential for a lot of computer vision tasks, like object detection and classification, development of self driving vehicles or virtual reality. In the context of

Figure 2.1: Example of semantic segmentation.

this project, semantic segmentation is the approach that makes the most sense, since we want to split images in regions containing obstacles, like sea bed, and background, such as areas containing only water.

There are several approaches to segment images, such as threshold based methods, edge detection, region based, clustering, watershed or using artificial neural networks designed for this purpose. [16]

2.2 Artificial Intelligence

The design and implementation of systems with close-to-human intelligence is, probably, the ultimate intellectual challenge. A system with such capacity must be able to possess and acquire knowledge, while being able to reason with it.

Artificial intelligence (AI) has seen many definitions over time. Winston [1984] describes AI as the study of algorithms that enable computers to behave in an intelligent way. Nowadays, whenever we open a book on AI it starts with a very cautious, yet understandable, definition of what AI is and what we can do with it. During its brief history, AI as seen periods of really high expectations followed by disappointment, much due to what opinion the media have about it. It is a subject that raises people's fear of subjugation to intelligent machines which lack human sensitivity and morals. However, one of the most common approaches to AI is to see human reasoning as a model instead of the end goal. Some accepted definitions are:

- Artificial Intelligence consists of creating machines that are able to mimic human behaviour. - Alan Turing
- "Artificial intelligence is that activity devoted to making machines intelligent, and intelligence is that quality that enables an entity to function appropriately and with foresight in its environment." - Nils J. Nilsson

- “Artificial inteligence is the science of making machines do things that would require intelligence if done by men” - Marvin Minsky

Since Alan Turing’s definition of AI around 1935, this field of computer science has seen tremendous progress. With the progress of other technologies, such as distributed computing, the evolution of computing power and the exponential growth of stored data, which is a very important fuel to the study of AI, we have seen this progress accelerate in the past few years. Today, we can see it in use in several fields.

Agriculture is one of the areas benefiting from AI, with increased yields by using agricultural robots and automated greenhouses with optimization techniques. In cybersecurity, AI and Natural Language Processing (NLP) are used to sort information into high or low risk intel and assess which attacks could be more harmful for organizations.[17] In finance, it is used for algorithmic trading (trading bots) or financial audits. [18] In military, it can be used for threat detection, predicting enemy positions, autonomous combat or surveillance vehicles. The latter is one of the fields that most invests in AI, with a worldwide spending of US$ 6.3 billion in 2020 for research and development. In healthcare, it can be used for initial classification and diagnosis based on CTs or MRIs, heart sound analysis or surgical robots. [19] With the COVID-19 pandemic, some companies developed AI-based products that would help predict where the virus would spread based on travel data and most affected regions. In services, we can see it put to use in human resources and recruiting, for resume screening or chat bots to automate repetitive communication. In marketing it can be used to analyse customer behaviour based on digital footprint, in order to use targeted marketing. In the transportation industry, with the development of self-driving vehicles. The list goes on and it is constantly increasing.

2.3 Deep Learning

When AI first started being studied, it tackled extremely well some problems that are hard for humans. Problems like hard calculations, following certain game rules or quickly draw information from large amounts of data, are easy for computers but not so easy for humans. Challenges and struggles appeared when it came to tasks that are intuitive for humans, such as recognising written and spoken language, sounds or objects in images. [6]

To address the abovementioned challenges, artificial intelligence researchers paved the way for a new field denominated Deep Learning. Most of the knowledge required to tackle tasks that are natural for humans is subjective and intuitive and, thus, hard to put into a formal set of rules. Deep learning can be defined as the deconstruction of complex input concepts into simple ones and the recombination to form a new complex output. This can be visualized as deep a multi layer graph. Each of these layers is formed by a set of neurons, hence the also used denominations Neural Networks or Artificial Neural Networks.

2.3.1 Neuron

Neurons are a core component of an Artificial Neural Network, since they make up the layers and are responsible for making critical calculations. Having a good understanding of how neurons work is key to better understand Neural Networks.

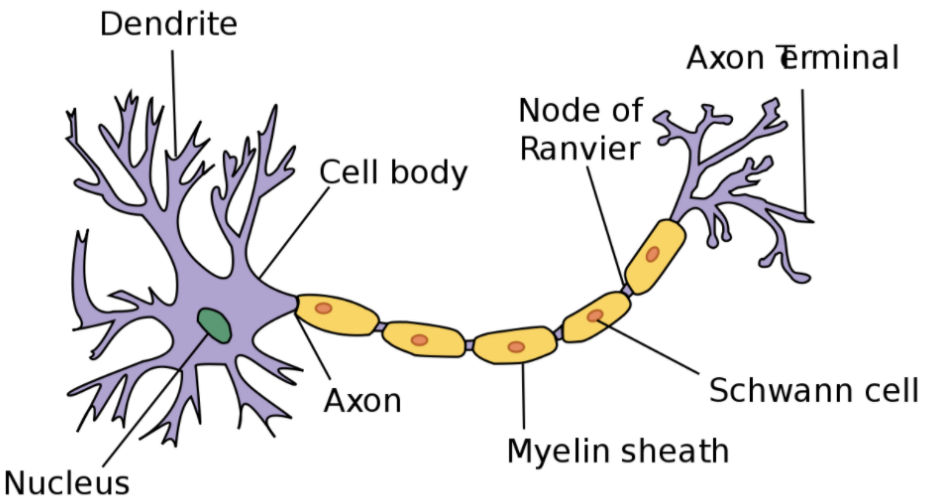


Figure 2.2: Biological neuron.

In fact, neurons in Deep Learning were inspired in biology. On a biological neuron, the dendrite of one neuron is connected to the axon of another through synapses. If we describe what a biological neuron does, in a narrow way, it receives one or more inputs, that can be provided by a dataset or another neuron at a previous layer, performs some calculations and a signal is sent to another neurons deeper in the network. In human brains, neurons are not all active at the same time and the same happens in Deep Learning models, neurons should be excited above a certain threshold in order to be activated.

On a Deep Learning neuron, each connection is represented by a weight w_i and each input is denoted x_i. So, the input for each neuron can be described as

$$i = w_1x_1 + w_2x_2 + ... + w_nx_n + b = \sum_{i=0}^{i=n}(w_ix_i) + b \tag{2.1}$$

or in matrix form

$$a = Wx + b \tag{2.2}$$

where W is a matrix containing all the weights that concern the neuron and b is a bias constant. The concept of weight is extremely important, since updating the weights is what makes training models possible. We will go through this concept later in this chapter.

The output o of each neuron is a non-linear transformation of its inputs and can be described as

$$o = \Phi(Wx + b) \tag{2.3}$$

where Φ is the activation function responsible for generating the signal that is going to be passed to the next neuron. Activation functions will be covered later in this chapter. The figure below (fig. 2.3) is a schematic of how an artificial neuron works.

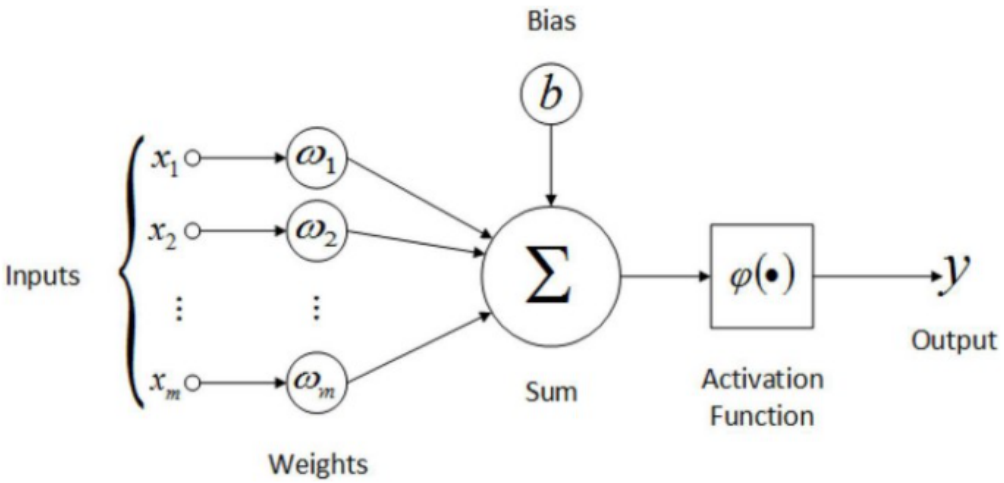


Figure 2.3: Neuron in Deep Learning Anagram. Source: [20]

2.3.2 Activation Function

An activation function is what defines the output of a neuron based on its inputs. Usually, activation functions perform non-linear transformations, which means that a change in the input may not lead to the same change in the output. Applying non-linear operations enables our model to fit better for complex data structures, compared to linear operations.

Choosing the best activation function is highly dependent on the objective of the project and design choices. For instance, *softmax* function works well on multi-class classification tasks or a hyperbolic tangent is mostly used for classification between two classes.

Name	1**Function**
Sigmoid	$\sigma(x) = \frac{1}{1+e^{-x}}$
Softmax	$\sigma(z)_i = \frac{e^{z_i}}{\sum_{z_j}^{K} e^{z_j}}$
Hyperbolic Tangent	$\sigma(x) = \frac{2}{1-e^{-2x}} - 1$
Rectified Linear Unit (ReLU)	$\sigma(x) = max(0, x)$

Table 2.1: Some activation functions. Source: [6]

The sigmoid and hyperbolic tangent functions receive an input $\mathbf{x}$ and perform non linear calculations to determine the output. For the softmax activation function, the input is a vector $\mathbf{z}$ of $\mathbf{K}$ elements and the output is a probability distribution over a finite set of classes. Regarding the ReLU, the function returns zero if it gets a negative input and returns $\mathbf{x}$ if it gets a positive one. These are some of the most used activation functions in Deep Learning.

2.3.3 Supervised Learning

Supervised learning is a deep learning paradigm which aims at creating functions that learn based on input-target pair examples.

In the learning phase, the function is presented with a training set, consisting of inputs and targets. Through each iteration, the error between the function's prediction and the target is computed, through a loss function. In the next iteration, the function's attributes (weights) are uptadated in a way that shall reduce the loss. The same steps are followed across a defined number o iterations. At the end of the training phase, the function should be able to generate accurate outputs from inputs different from those in the training set. Artificial Neural Networks are an example of machine learning algorithm that follows the supervised learning paradigm.

2.3.4 Loss Function

The concept of loss function is widely used in computer science. In the field of DL, the role of this function is to measure the difference between the result at the output layer of the model and the corresponding target that the model was supposed to reach. Our goal in DL is to minimize this function, hence having the model's prediction as close as possible to the target.

Each DL task has very unique specifications. Every loss function has different geometry, therefore with inappropriate loss functions and poor initialization it becomes hard to find the configuration of weights that minimizes loss. Since this thesis covers a segmentation problem, that can be viewed as a classification one, loss functions for classification will be covered.

Binary Cross Entropy

In DL, classification aims to maximize the probability of a certain object belonging to a category. In the field of this thesis, the goal is to evaluate whether a pixel is foreground or background, so our problem becomes a binary one.

Binary Cross Entropy is commonly used for binary classification problems. The function can be written as

$$L_{BCE}(x, y) = -\frac{1}{m}\sum_{i=1}^{m} y_i \cdot log(x_i) + (1 - y_i) \cdot log(1 - x_i) \tag{2.4}$$

where $\mathbf{x_i}$ is the i-th model output and y_i is the corresponding target. This function tells us how far the prediction is from the target. In this case, the prediction is measured, for each pixel, as a probability of belonging to a given category. The goal is to minimize the loss function, making the prediction as accurate as possible.

2.3.5 Gradient Descent

The gradient descent is a very important concept in Machine Learning in general and Deep Learning in particular, since, according to [6], it powers most of the algorithms. The goal of gradient descent

algorithms is to minimize the error across training examples, thus it is responsible for fitting a model to the data structure.

For better understanding, we can simplify the problem and visualize how to minimize a squared error on a neuron with only two inputs and two weights, w_1 and w_2. Then, we can think of a tri-dimensional space, where the horizontal dimensions correspond to the two weights and the vertical axis is the loss function. If we consider the errors for all possible weight configurations, we get a surface that can be visualized in figure 2.4.[21]

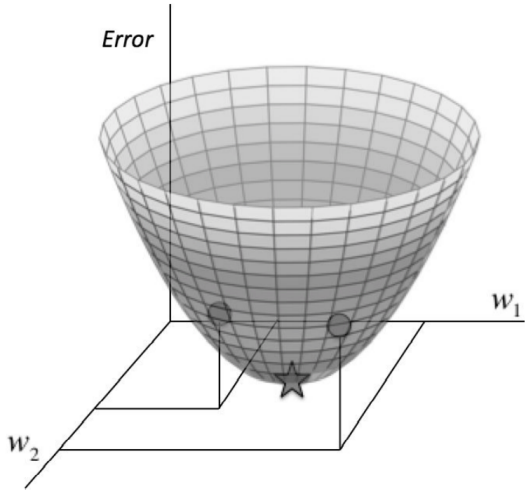


Figure 2.4: Quadratic error surface. Source:[21]

Another way to visualize the figure 2.4 is as series of elliptical contours. In a same elliptical contour, different configurations of w_1 and w_2 yield the same error. The closer the contours are to each other, the steeper the slope and smaller the error. So, our goal is to get as closer as possible to the center of the ellipses. [21]

Now let's imagine we randomly initialize the weights and we can compute. At this point we can already define the error and we find ourselves somewhere in the quadratic error surface. To minimize our error, the weights need to be updated in a way that consecutively yields lower error estimates. To do this, we compute the gradient of the error with respect to the weights for our current position and find the direction of the steepest descent. Now, we find ourselves at a different position, which should be closer to the minimum than the previous one. We can take another step towards minizing the error by following the same procedure. We keep doing this until we converge towards a minimum.

Stochastic Gradient Descent

As the training set grows, the complexity of the model grows along, thus taking a single gradient step becomes computationally expensive to the extent it becomes prohibitive.

The loss function used by machine learning algorithms can be written as sum over training examples of individual per-example losses. For a dataset with m examples, we have

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} L(x_i, y_i, \theta) \tag{2.5}$$

where $\mathbf{L}$ is the per-example loss, $\mathbf{x_i}$ is the input, $\mathbf{y_i}$ is the associated target and θ is the vector of weights.

For these cost functions, the gradient descent requires the calculation of

$$\nabla_\theta J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta L(x_i, y_i, \theta) \tag{2.6}$$

and this is where computational cost becomes a concern. As the dataset grows to millions or billions of examples, the time to compute each step becomes inefficiently large.

The solution is to estimate the gradient using a subset drawn uniformally from the training set with size $m' < m$. The gradient estimate becomes

$$g = \frac{1}{m'} \sum_{i=1}^{m'} \nabla_\theta L(x_i, y_i, \theta). \tag{2.7}$$

As per [6] as the training set size m increases, the batch size m' can stay fixed, as it is enough to use hundreds of examples to estimate a gradient for a training set with millions or billions of examples.

The stochastic gradient descent algorithm then follows the step

$$\theta \leftarrow \theta - \epsilon g \tag{2.8}$$

where ϵ is the learning rate and θ is a vector of weights.

In the past, gradient descent was regarded as slow and inefficient solution. Today, with the evolution of computational power, we know that machine learning algorithms work well with this approach. It may not guarantee that we arrive to a minimum, but is often quick to find an error value that is low enough to be reliable.

RMSProp

The RMSProp algorithm, standing for Root Mean Square Propagation, is a gradient based optimization technique proposed by Geoffrey Hinton in 2012. Some optimization algorithms may find trouble converging to an optimal solution when they find themselves in a local minimum, a convex bowl, of a non-convex function. RMSProp solves this by using an exponentially decaying average to discard values from an extreme past, as if the optimization algorithm was initialized inside that bowl, making it converge fast. [6]

This algorithm has shown effectiveness in optimizing deep neural networks, being one of the preferred algorithms by deep learning researchers and users.

The algorithm starts by sampling a minibatch of m samples from the training set $\{x_1, ..., x_m\}$ with the corresponding targets $\{y_1, ..., y_m\}$.

The accumulated square gradient is initialized as zero $r = 0$ and the gradient is computed

$$g = \frac{1}{m'} \sum_{i=1}^{m'} \nabla_\theta L(x_i, y_i, \theta). \tag{2.9}$$

followed by the accumulated square gradient

$$r = \rho r + (1 - \rho) g \circ g. \tag{2.10}$$

where $\circ$ denotes the Hadamard product. The next step is to compute the weight update

$$\Delta\theta = -\frac{\epsilon}{\sqrt{\delta + r}} \circ g. \tag{2.11}$$

with δ being a small constant used to prevent the algorithm from exploding when dividing by small numbers, usually takes the value of 10^{-6}. Now the weights can be updated by doing

$$\theta \leftarrow \theta - \epsilon g. \tag{2.12}$$

2.3.6 Artificial Neural Networks

An Artificial Neural Network (ANN) is a collection of connected neurons. The connected neurons form layers that can be split into input layer, which receives the inputs from the dataset, the hidden layers and the output layer, that yields the result after running and example through the model.

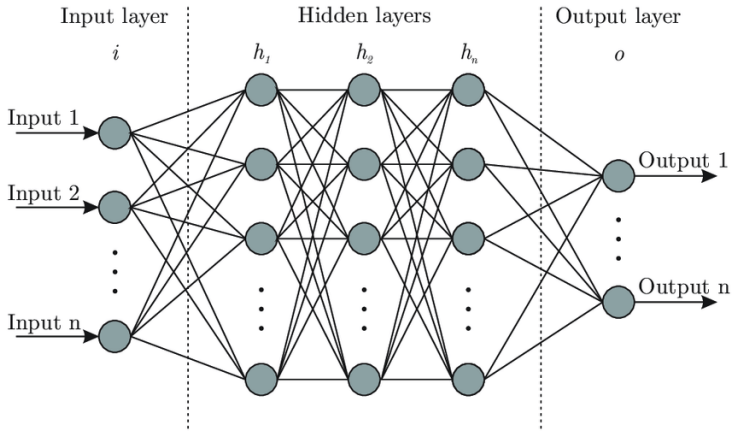


Figure 2.5: Artificial Neural Network. Source: [22]

Neurons are connected to each other accross layers, meaning that neurons from one layer are connected to neurons in the next one. The connections between two neurons are called edges and are associated to a weight. In other words, the signal that is sent by the activation function of a neuron is multiplied by the weight of the edge connecting this neuron to the next one. The weights are adjusted during the training phase and are responsible for giving more or less importance to a signal coming from a neuron. Layers can be connected to each other in different ways and perform different transformations to their inputs.

Backpropagation

Usually, the term backpropagation is misunderstood as meaning the whole learning algorithm of neural networks. In fact, back propagation refers only to the method used for computing the gradient,

while another algorithm, like stochastic gradient descent, is used to perform the learning having the gradient. When working with learning algorithms the gradient that is often required is the gradient of the error function with respect to the network parameters, the weights. The back propagation algorithm can be applied not only on the learning process, but also when analyzing the trained model. In fact, the idea of computing derivatives by propagating information through the network's layers can be useful for general mathematical applications, such as calculating gradients of a function with multiple outputs. [6]

If we consider an artificial neural network with N layers, let W_n be the weight and F_n the activation function of the n_{th} layer. We can describe the output as

$$X_n = F_n(W_n \cdot X_{n-1}) \tag{2.13}$$

where X_{n-1} is the output of the previous layers and thus the input to the current layer. If we define the loss function L that compares the outputs with the targets Y, we can describe the error as

$$E = L(Y, X_n, W_n). \tag{2.14}$$

If we differentiate with respect to the weight

$$\frac{\partial E}{\partial W_n} = \frac{\partial E}{\partial X_n} \cdot \frac{\partial X_n}{\partial W_n} \tag{2.15}$$

$$\frac{\partial E}{\partial W_n} = \frac{\partial E}{\partial X_n} \cdot \frac{\partial F_n(W_n \cdot X_{n-1})}{\partial w_n} \tag{2.16}$$

Now that we have $\frac{\partial E}{\partial W_n}$, we could proceed to the actual learning process of updating the weights. In the simplest way, the weights of the n_{th} layer would be updated by an amount proportional to the computed gradient

$$\Delta W_n = -\epsilon \cdot \frac{\partial E}{\partial W_n} \tag{2.17}$$

where ϵ is the learning rate. [23]

2.3.7 Convolutional Neural Networks

Convolutional neural networks (CNNs) are a special kind of artificial neural network used for tasks where data structures can be seen as grids. For example a time series data or image data. The last can be seen as 2D grid of pixels. The name 'Convolutional' is due to the fact that at least one layer of this family of networks performs a **convolution** operation instead of general matrix multiplication. In Deep Learning, the convolution operation can be done in different ways and may differ from the definition of convolution in other fields, such as mathematics. Besides convolution, most CNNs also execute an operation called **pooling**, both will be covered in this section. [6]

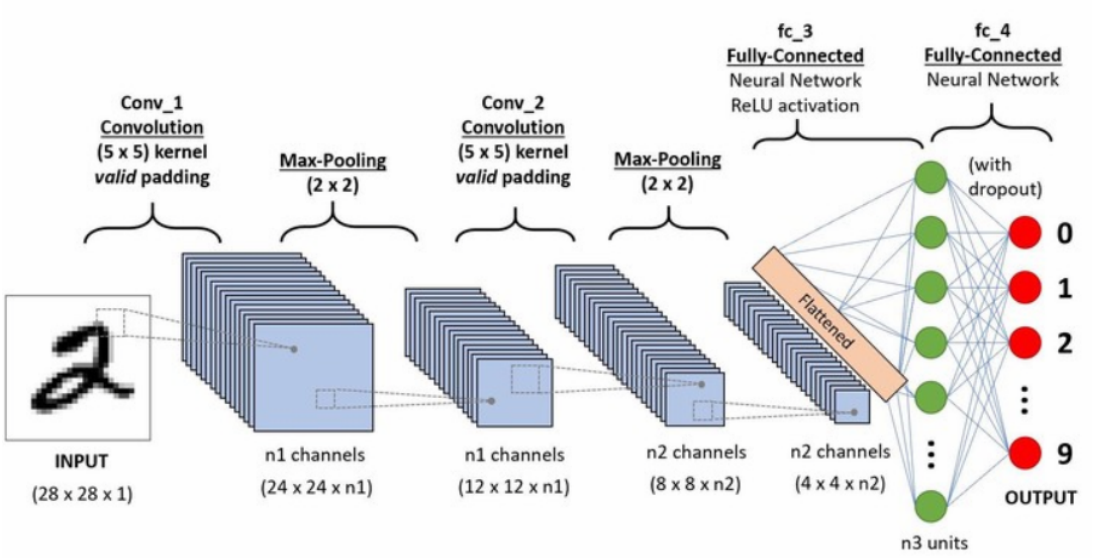


Figure 2.6: Convolutional Neural Network. Source: [24]

In the architecture of figure 2.6 the input layer will store pixel values of the image. The convolutional layer is responsible for determining the output of neurons connected to certain regions of the input layer, by performing the scalar product of connection weights and pixel values. The pooling layer downsamples the output of the convolutional layer to reduce computational cost and improve training and execution times. The fully connected layers work as in other ANN architectures, aiming to produce class scores for classification.

Convolution

At the output of the convolutional layer new images, called feature maps, are generated. The feature map is a version of the original image where unique features are emphasised. This layer also has a unique feature compared to layers in other neural networks, since weights are not present in connections between neurons, instead there are filters (*kernels*) responsible for the convolution where weights are stored.

Since the convolution operation occurs in a 2D plane, it is better explained graphically. Therefore, figure 2.7 is a good representation of the operation. First, we perform the dot product between the kernel and a submatrix of the input matrix, the value is stored in a new entrance of a new matrix (feature map). Then we do the same for the next submatrix of the input image. It should be noted that kernels usually come in the form of 3×3 or 5×5 matrices and the weights are not selected by the network designer, instead they are usually randomly initialized and iteratively updated through the training process.

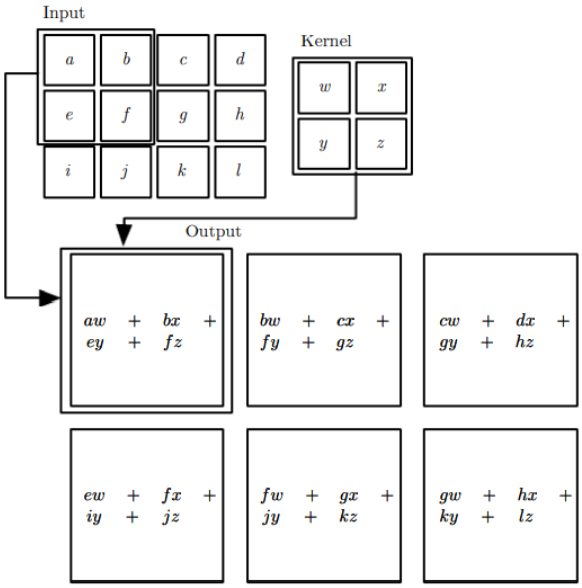


Figure 2.7: Graphic representation of a convolution. The kernel slides through the input image, performing dot products between the kernel values and a submatrix of the input matrix. Source: [6]

A convolutional layer usually has several kernels that create different feature maps. Each feature map goes through an activation function before the output of the layer is yielded. Although this layer has different characteristics from the ones usually used in Deep Learning, the activation function is identical. The ReLU activation function is used in most cases, but there are implementations where sigmoid and hyperbolic tangent are used. [25]

Pooling

The pooling layer reduces the size of the image, in order to decrease computational cost and training times, by combining neighboring pixels into one representative value.

The first stage of pooling consists of choosing how the pixels to be combined should be selected and how to combine the values. The number of pixels to be combined depends on the problem being solved and they are usually combined by computing the average of the selected pixel values or choosing the one with highest intensity.

The pooling operation is very simple and since it is a matrix operation it is best explained by a scheme. The figure 2.8 illustrates max and mean pooling.

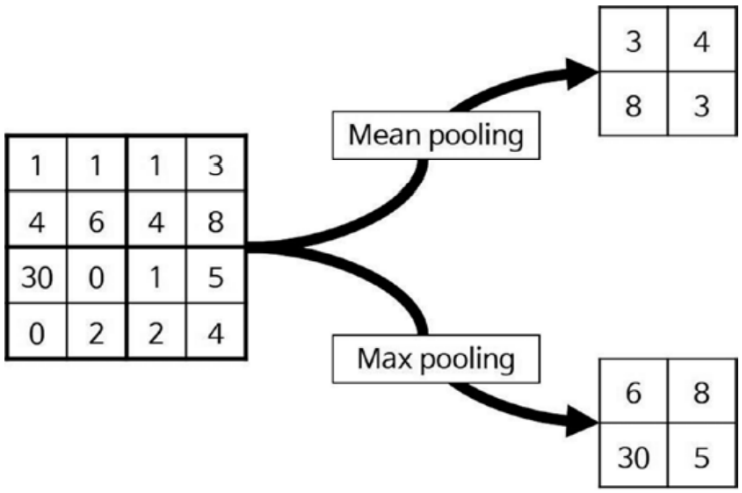
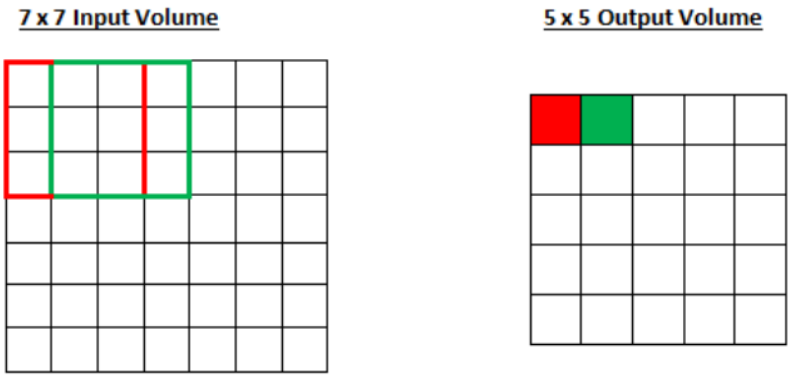


Figure 2.8: Graphic representation of the pooling operation. Mean pooling operation performs the mean of a submatrix of the input image. The max pooling operation takes the maximum value of a submatrix of the input image. Source: [25]

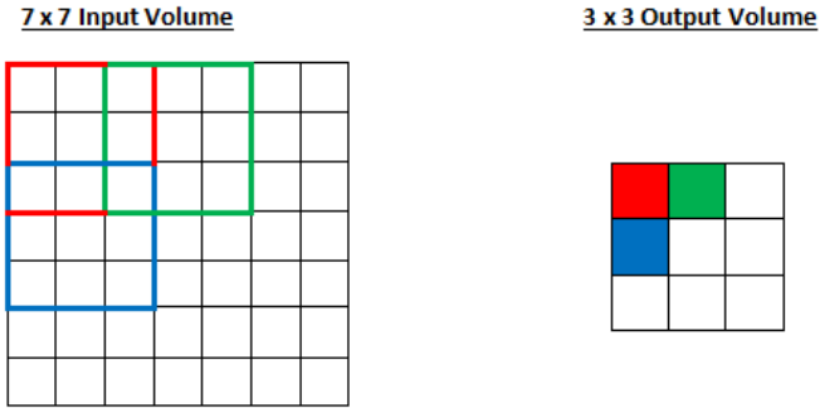
In fact, pooling is another convolution. The differences from the convolution layer reside in the fact that the convolution filter is stationary and convolution areas do not overlap. Apart from improving computational efficiency, pooling may also improve classification accuracy, by eliminating areas where no object of interest is present, like in an image where the obstacle is not centered.

Stride

Stride is a concept of convolutional neural networks and is of great importance for the convolution operations. This parameter determines how the kernels move through the image. If stride is set to one, the convolution kernel will slide 1 pixel to the right after performing the first convolution and the same happens to the following operations. Stride is usually set so that the output is an integer, which implies setting this parameter to an integer value. Stride is also a form of reducing the ammount of features to be passed to the following layers, as bigger stride will lead to smaller outputs.



(a) Convolution with stride 1. The 3×3 kernel slides one pixel to the right after performing the dot products of the first convolution.



(b) Convolution with stride 2. The 3×3 kernel slides two pixels after performing the dot products of the first convolution.

Figure 2.9: Comparison between stride 1 and 2. Stride is also a way to reduce the ammount of features to be processed, as bigger stride yields smaller outputs. Source: [26]

Padding

To determine the output size of a convolutional layer one can use the following expression

$$O = \frac{W - K - 2P}{S} + 1 \tag{2.18}$$

Where **O** is the output size, **W** os the input image's height or width, **K** is the kernel size, **P** is the padding and **S** is the stride.

As data goes through several convolutional layers, the output volumes will keep decreasing. In the first layers of CNNs it is usually best to preserve as much information as possible about the original input image. Imagine if we have a 32×32 image and the output should have the same size. To do this we can apply zero padding, or simply padding, which consists of padding the input image with zeros around the borders, as can be seen in figure 2.10. Zero padding does not affect the values of the convolutions and it aids in preserving features.

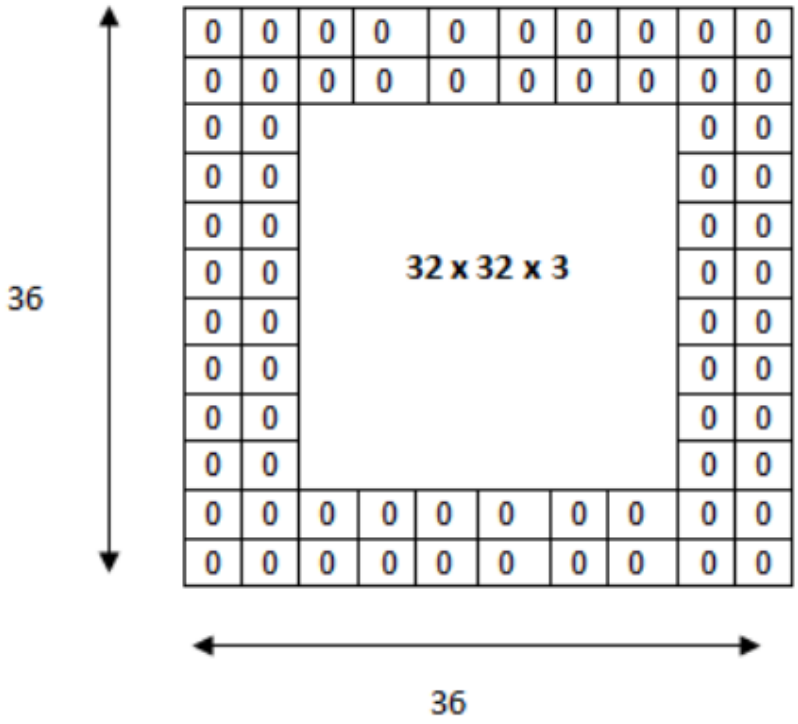


Figure 2.10: Zero padding of size two. Source: [26]

2.3.8 VGG16 Network

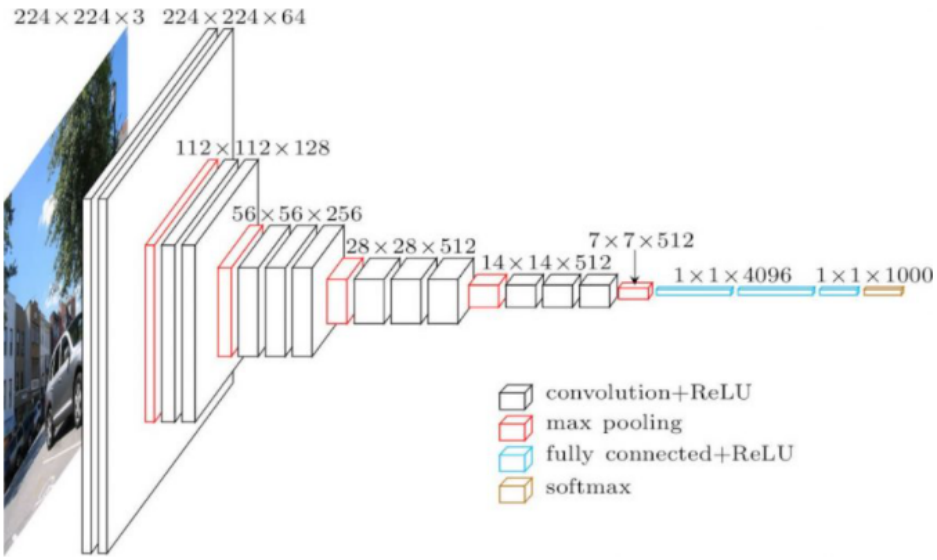


Figure 2.11: Architecture of the VGG16 Convolutional Neural Network. Source: [27]

The VGG16 [8] is a deep convolutional neural network mostly used for image classification tasks. It was the winner 2014 Large Scale Visual Recognition Challenge and is still considered an excellent vision model.

The inputs are fixed sized $224 \times 224 \times 3$ images, in RGB format. The image goes through a series

of convolutional layers with 3×3 filters. Convolution stride is fixed to 1. This architecture also employs 5 max-pooling layers following some of the convolutional layers, with 2×2 filters with stride 2. The convolutional and pooling layers are followed by 3 fully connected layers, the first two have 4096 channels and the last has 1000 channels. Each of the channels of the last fully connected layer concerns one class of the classification problem. The final layer, responsible for the output classifications, is a soft-max layer. The activation function for all the hidden layers is the Rectified Linear Unit (ReLU). [8]

Due to the high number of layers it employs and the number of weights used, this architecture is extremely slow in the training process. Another disadvantage is the fixed size inputs, that should always be 224×224 images.

2.3.9 Fully Convolutional Networks

Fully convolutional networks [12] are neural network architectures mostly used for semantic segmentation tasks.

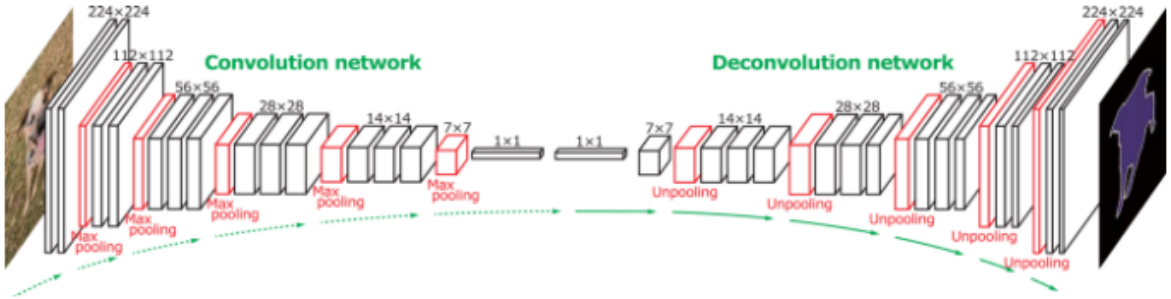


Figure 2.12: Fully convolutional network. Source: [28]

Today, a good amount of the best performing state-of-the-art semantic segmentation methods relly on fully convolutional networks. These networks are able to yield dense pixelwise predictions on arbitrary sized inputs and can be trained end-to-end. Another advantage is that these networks are built on top of CNNs which are able to produce rich feature representations. These CNNs can be pretrained models on general purpose datasets, further reducing the time required to train the network.

These networks are denominated 'fully convolutional' because they do not have a fully connected layer like CNNs and most artificial neural network architectures. Fully connected layers were removed for two reasons: first, they entail a single input size, not allowing the model to perform on arbitrary sized inputs; second, they only yield a single feature vector for the entire image. To replace the fully connected layer fully convolutional layers were introduced, which are capable of returning a feature vector for each pixel in the image. This capability allows for pixel-wise classification, which ultimately can be used to create semantic segmentation masks, by upsampling these feature maps in a process called deconvolution.

From classifier to segmentation

The work of Long et. al [12] takes networks used for image classification problems, like GoogleNet, AlexNet or VGG16. In all the previous networks, the classifier is discarded and all the fully connected layers are converted to convolutions with 1×1 kernels. Convolutional layers with size 1 kernels are similar to fully connected layers but are capable of dealing with arbitrary size inputs. In a normal CNN, with fully connected layers, at the end of the classifier, the output would be a single predicted label. But since the fully connected layers are replaced by convolutional layers, what the network outputs is a feature map where features are assigned to a certain class.

The output of the fully convolutional layers has lower resolution than the input image, due to the pooling layers used to downsample. The method used by [12] uses transposed convolutions, or deconvolutions, to upsample the predicted feature map until the size of the input image is reached.

From training and validating this architecture on the PASCAL dataset, Long et al [12] found the FCN with VGG16 was the one that yielded the best results.

Deconvolution

Deconvolution, or transposed convolution as can be seen in some literature, is a mathematical operation that reverses the effect of a convolution. The input feature map is upsampled to a desired output feature map. In the case of a FCN, it is usually upsampled until the dimensions of the input image are reached.

As in CNNs, the deconvolution kernels do not have fixed values. Instead, they are iteratively updated during the training phase, through the backpropagation of the loss computed at the output of the network.

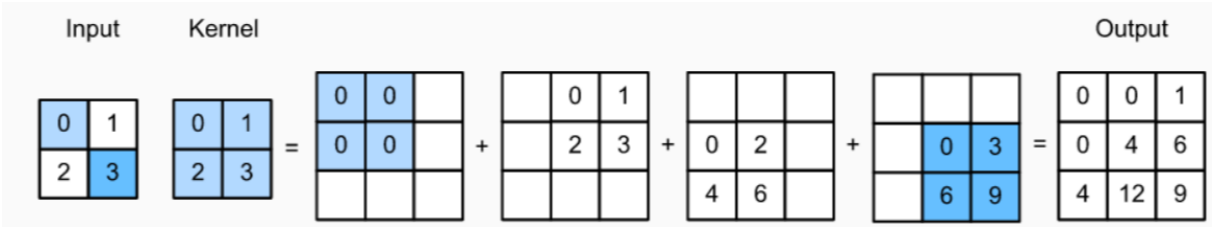


Figure 2.13: Deconvolution process schematic. Deconvolutions are used to upsample the input feature map to a desired output feature map using learnable parameters. Source: [29]

Figure 2.13 portrays the deconvolution process. An input matrix and a kernel both of size 2×2 are considered for the example. The first step is to take every element of the kernel and multiply for every element of the input image, creating the 4 matrices displayed in the image. Next, all the matrices are added giving place to a 3×3 matrix, which has larger dimensions than the input.

2.3.10 Network Training Parameters

Epochs

The number of epochs is a hyperparameter that sets the number of times the algorithm will iterate over the entire dataset. One epoch means that each data instance has ran through the network once, updating the values of the weights.

Batch Size

The batch size defines the number of data samples that are ran through the network before updating the weights. A training dataset can be split into one or more batches. If the batch size matches the size of the dataset, the learning process is called batch gradient descent. If the batch size is just 1 sample, we call it stochastic gradient descent. If the batch size is greater than 1 and less than the size of the full dataset, it is called mini-batch gradient descent.

Learning Rate

The learning rate is a parameter that determines how much the weights should change in response to the determined loss. Choosing the correct learning rate is crucial, as a value that is too small can lead to extremely long training times, whereas a value that is too high may cause the model to learn non-optimal weight configurations too fast.

Step Size

The step size is a parameter that sets the weight variation from one epoch to another. It depends on the loss function **E**, epoch **t** and learning rate η as follows

$$\Delta w = w(t+1) - w(t) = -\eta \frac{\partial E(w)}{\partial w}. \tag{2.19}$$

Weight Decay

Weight decay is a network hyperparameter used in a regularization technique, that aims to prevent overfitting and keep the weights small, to prevent weights from growing out of control and avoid exploding gradients.

The regularization is done by adding a penalty to the loss function, that is the L2 norm of all the network weights multiplied by the weight decay parameter. Meaning that the loss **L** takes the following form

$$L = L + W_D \cdot \sqrt{\sum_{k=1}^{n} |w_k|^2} \tag{2.20}$$

where $\mathbf{W_D}$ is the weight decay and w_k are the weights.

Momentum

Momentum is a parameter that plays a role in the optimization process. Sometimes refered to Stochastic Gradient Descent with momentum is a method that helps accelerate gradient vectors in the right directions and leads to faster convergence.

Adding momentum can be seen as a moving average over a series of noisy data. The moving average allows for more clear reading of the data. This can be applied to training a neural network if we average over our gradients. The weight updating algorithm can be written as

$$V_t = \beta V_{t-1} + (1-\beta)\nabla_w L(W, X, y) \tag{2.21}$$

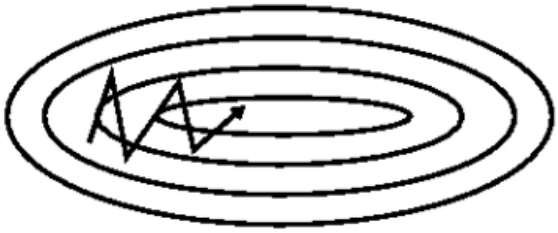
$$W = W - \alpha V_t \tag{2.22}$$

where **W** are the weights, **L** is the loss function, β is a parameter that ranges $[0; 1]$, α is the learning rate and **V** is an average of the gradients. Note that if $\beta = 0$, **V** will be just the gradient of the loss function.

In classic Stochastic Gradient Descent we are not computing the exact gradient of the loss function, we are estimating it on a smaller batch of data. Meaning that we will be getting noisy derivatives and not moving in optimal directions all the time. Weighted averages like that on equation 2.21 provide better estimates for the derivatives because it takes into consideration more data. It uses the gradient of the loss function as well as the gradient from a time instant $t - 1$.

Another reason for momentum to be benefitial is the fact gradient curves have ravines. Ravine is an area where the curves are steeper in one direction than the others, they are common when closer to minimum and classic stochastic gradient descent has some difficulties converging to optimal minimum in these conditions. Momentum helps SGD converge in the fastest way.

(a) SGD without momentum



(b) SGD with momentum

Figure 2.14: Stochastic gradient descent without momentum (a) and with momentum (b). Source:[30]

2.3.11 Transfer Learning

The field of machine learning has been used with success in applications where the algorithms draw patterns from training data in order generate predictions with new data. In the most traditional practices of machine learning, the new data, used to predict future outcomes, has a very similar feature space to the training data. When the two datasets are not similar, the algorithms tend to perform poorly. When solving certain problems obtaining a sufficient amount of training data to train a model to achieve good results on testing data can be very hard and expensive. This is the motivation for transfer learning, developing a model that is able to operate on a target domain after being trained on a related source domain.

Transfer learning aims at improving the accuracy of a learner by transferring information from a related knowledge space. Drawing an example, consider we want to develop a model to predict customer sentiment from product reviews. We make the assumption that there is high availability of data from camera reviews, which we will use to train the model. If the model was to be used on camera reviews, it is expected to perform well, since machine learning algorithms thrive in these conditions. But if we want to apply this model to food reviews, then we should expect the results to degrade, since there are differences between the two domains. However, the two domains still have common characteristics, like the type of language a customer uses when happy or unhappy. Thus, in the absence of sufficient data in the target domain, we could use the related camera reviews domain to improve the accuracy of our model. [31]

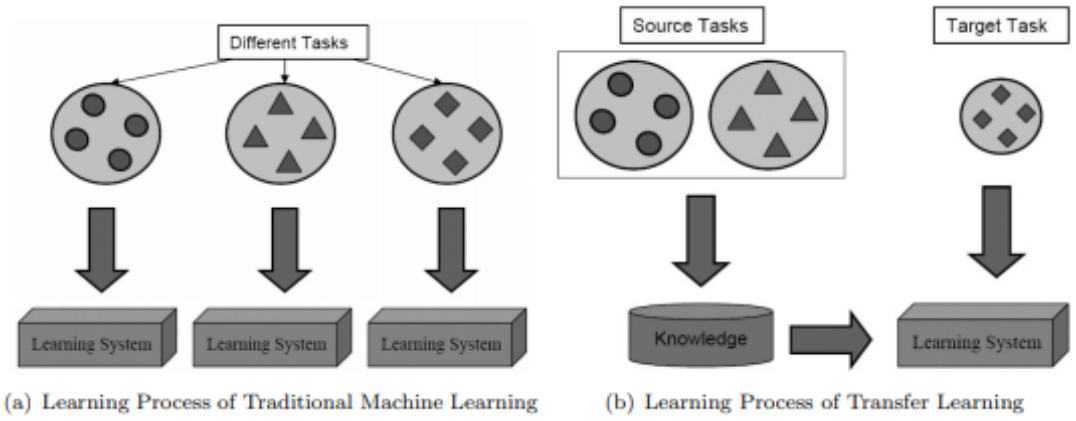


Figure 2.15: Comparison between transfer learning and traditional machine learning. In transfer learning a knowledge base previously trained for general tasks is seized for more specific tasks. Source: [32]

2.4 Simultaneous Localization and Mapping (SLAM)

Simultaneous Localization and Mapping (SLAM) is a computational problem of reconstructing and updating a vehicle's trajectory by mapping while, simultaneously, knowing the location of the vehicle in the same map. The SLAM problem is widely regarded as one of the major problems in the field of fully autonomous vehicles [33] and although the field has seen major progresses, it is still a great challenge, particularly when a vehicle is intended to navigate through dynamic large scale environments, such as

deep ocean or space exploration. SLAM algorithms are commonly used in self-driving cars, autonomous underwater vehicles, unmanned air vehicles and robotics.

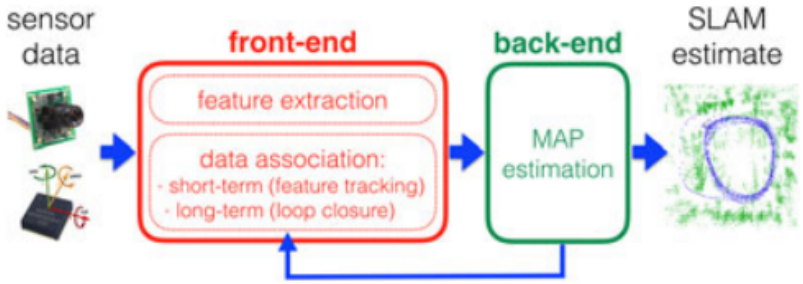


Figure 2.16: Architecture of a SLAM system. The front end is responsible for abstracting sensor data into a model suited for estimation. The back end processes the abstracted data from the front end, generating a SLAM estimation, with the trajectory of the vehicle. Source: [34]

2.4.1 Definition of the SLAM Problem

The SLAM problem can be described as a robot, capable of sensing its surroundings, roaming a previously unknown environment with known starting coordinates. The way the robot moves is not certain, which increases the difficulty of determining its global coordinates. SLAM algorithms should deal with the task of mapping the environment while simultaneously knowing the robot's position in this map.

For better understanding, terminology from the field of probability will be used. For robots moving in the ground, the vehicle's location is usually a three-dimensional vector with two coordinates for its position in the plane and one value for the robot's heading. The trajectory followed by the robot is given by

$$X_T = \{x_0, x_1, x_2, ..., x_T\} \tag{2.23}$$

where each x_i is a known location of the robot and T denotes a terminal time period.

Odometry is a robotics technique where data from motion sensors is used to estimate changes in position over time. With u_t being the odometry that describes the changes in position between two intervals, the sequence

$$U_T = \{u_0, u_1, u_2, ..., u_T\} \tag{2.24}$$

can describe a series of steps in the robot's motion. This data can be retrieved from controls given to the vehicle's motor, sensors or cameras. If the measurements of U_T were noise free, then it would be sufficient to retrieve the trajectory X_T.

The robot can sense the environment via cameras or sensors. The environment, composed by objects, landmarks, floor and other surfaces, is usually assumed to be static. Let m denote the ground-truth map of the environment. The measurements from the robot's sensor provides information that relates the map m with the robot's position x_t. If the robot measures exactly once in every time instant,

the set of measurements is described by

$$Z_T = \{z_0, z_1, z_2, ..., z_T\}. \tag{2.25}$$

Now the SLAM problem can be resumed to generating a map m of the location and recover the robot's trajectory X_T from the sensor's measurements Z_T and the odometry data U_T. Current literature distinguishes between two major SLAM problems: online SLAM and offline SLAM.

Online SLAM focuses on recovering the current robot position x_t and it can be mathematically described as

$$p(x_t, m|Z_T, U_T). \tag{2.26}$$

Offline SLAM, sometimes called full SLAM, seeks to determine the full path of the vehicle, as well as the map, after observation. Mathematically the problem is defined as

$$p(X_T, m|Z_T, U_T).$$

Both SLAM problems are very important. Offline SLAM can be used for generating a map and recover textures and structures after exploring a certain area with a robot, while online SLAM is used for real time map and position estimation. In terms of algorithms, the two differ in one major point: for online SLAM, algorithms must be able to process one data item at a time for real time accuracy; for offline SLAM, since the algorithm is ran post-exploration with the acquired data, the methods usually work in batches, being able to process all the data at the same time.

Furthermore, to tackle both problems, two more models should be considered. One that sets a relation between odometry measurements and the robot's positions

$$p(x_t|x_{t-1}, u_t), \tag{2.27}$$

and another that relates the sensor's measurments of the surroundings with the map and the robot's location

$$p(z_t|x_t, m). \tag{2.28}$$

In the last we have the problem of not knowing neither the robot's position x_t nor the environment's map m. The Bayes rule allows us to transform the mathematics of equation 2.28 into an equation where we can retrieve probability distributions over the map and position

$$p(z_t|x_t, m) = \frac{p(x_t, m|z_t) \cdot p(z_t)}{p(x_t, m)}. \tag{2.29}$$

2.4.2 ORB-SLAM

ORB-SLAM [35] is a real time feature-based monocular visual SLAM algorithm. Visual SLAM aims at reconstructing the camera's trajectory while mapping the environment. In order to efficiently provide accurate representations of trajectory and the explored environment, an online SLAM algorithm should

be able to address the following tasks:

- In a set of keyframes, which are frames that set the beginning or end of a transition, the observed features should correspond.
- As the number of keyframes increases, the algorithm should avoid redundancy when selecting those frames.
- A set of keyframes and observed points with significant loop closure matches, which is the act of determining if a vehicle has returned to a previously visited area.
- An initial estimate of keyframe and feature positions for the optimization process.
- Create a real-time local map that allows for scalability as the amount of data increases.
- The ability to perform real time optimization for loop closure.

ORB-SLAM uses bundle adjustment (BA) to perform all the above tasks. BA is a non-linear least-squares optimization problem that amounts to refining a series of initial camera and trajectory parameters to estimate the set of values that describes the locations of observed features more accurately.

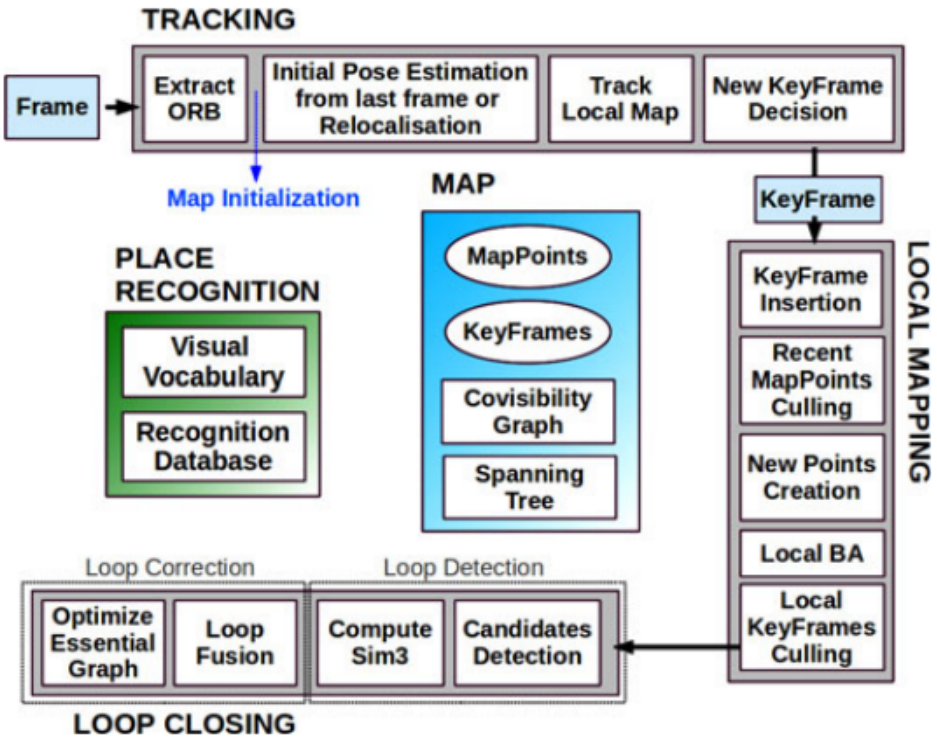


Figure 2.17: Schematic of ORB-SLAM system, with all the steps performed for each of the three tasks: feature tracking, local mapping and loop closing.

In terms of feature detection and choice, ORB-SLAM uses the ORB feature detector, which allows for fast computation while maintaining accurate results. One of the core ideas of ORB-SLAM is that the same features are used for very different tasks, such as mapping, tracking, place recognition and loop closing, contributing to overall efficiency of the algorithm.

Figure 2.17 shows the architecture of this SLAM algorithm, it exploits the use of three threads running simultaneously in parallel: one for tracking, another for local mapping and the last for loop closing. The tracking is responsible for estimating the camera position and infer when a new frame is a keyframe. To do this, the first stage is a feature matching between the new frame and the previous one for optimizing the camera pose estimation with bundle adjustment. If features do not match, a global relocalization is performed. Having an initial pose estimation, the local map is retrieved from keyframe features. The algorithm continues to search for matching features using reprojection and the camera pose is continuously optimized with those matches. The tracking part of the system is also responsible for determining if a certain frame is a new keyframe.

The local mapping thread takes the new keyframes and performs bundle adjustment to attain a reconstruction of the area surrounding the camera. Also, local mapping searches for correspondence between features in the new keyframe among the other connected keyframes. As tracked points keep increasing, the algorithm performs a point culling policy in order to keep only the highest quality points. The same is done to keyframes.

The last thread, loop closing, searches for loops everytime a new keyframe is inserted. To ensure consistency, pose graph optimization is performed. This thread is also responsible for finding and fusing duplicated points to avoid redundancy.

To perform all optimizations, ORB-SLAM uses the Levenberg-Marquardt algorithm.

Each tracked map point stores valuable information, such as:

- Its 3-D coordinates in the map coordinate system
- The viewing direction. A viewing direction is a ray that joins the optical center of a keyframe to the point.
- An ORB discriptor, which relates information between all the keyframes where that point is observed.
- The maximum and minimum distances at which that point can be observed.

The keyframes also store the following information

- The camera pose.
- Intrinsic characteristics of the camera, such as focal distance.
- All the features extracted by ORB from that frame.

ORB-SLAM uses covisibility graphs to represent keyframes. In these graphs each keyframe is a node and edges connecting different keyframes mean there is shared observation of map points. Each edge has weight that is equal to number of shared map points.

Whenever a loop is detected and closed, it shall be corrected using pose graph optimization. As complexity increases with the number of keyframes, the covisibility graph can become very dense, therefore the pose graph optimization is performed on a smaller *Essential* graph that keeps all the keyframes but

uses less edges, which keeps accurate results at lower complexity levels. Every new keyframe is inserted into the graph linked to another keyframe which shares the most point observations. If the culling policy determines a certain keyframe should be removed the algorithm updates the edges affected by the keyframe. The essential graph contains all the keyframes, but only the edges from the covisibility graph with high weights are kept (e.g. weight greater than 100) and also the edges that mark loop closures, which results in a network of of highly important data, adequate to achieve accurate results. A graphical representation of a scene reconstruction with the observed map points, keyframes and the generated covisibility and essential graphs can be seen in figure 2.18.

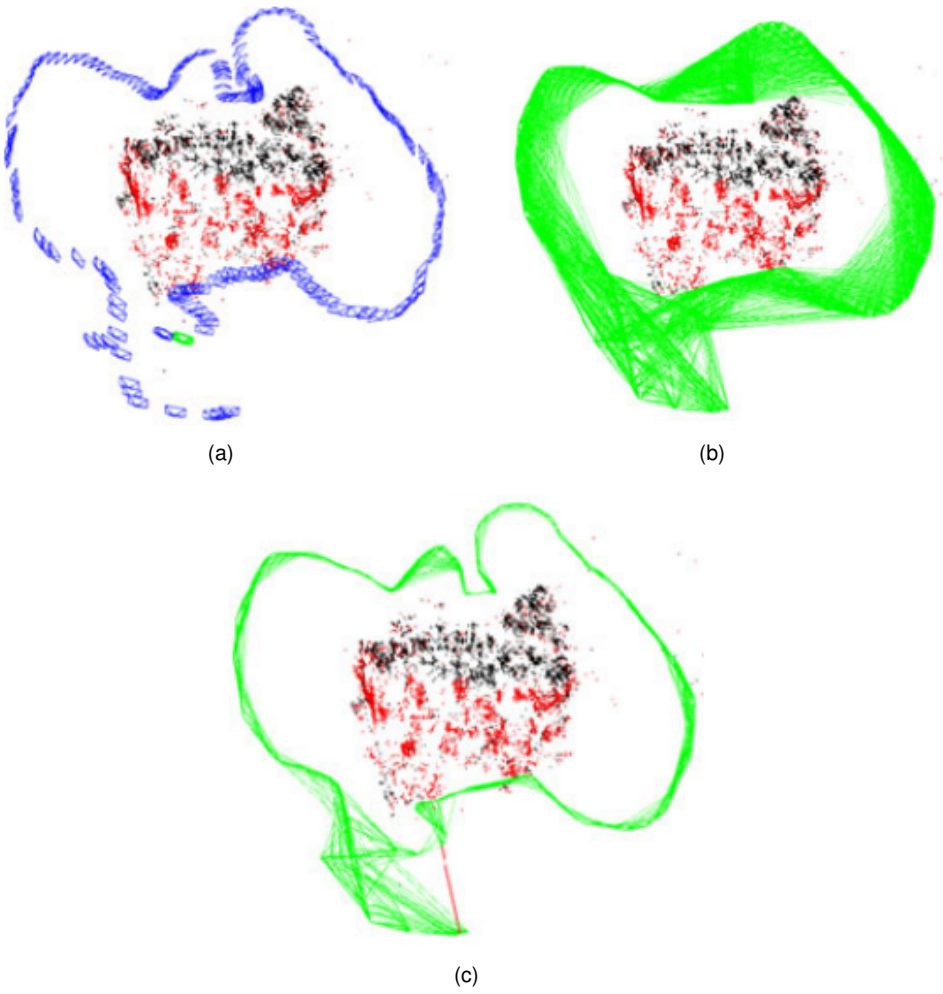


Figure 2.18: **(a)** Keyframes (blue), map points (red and black), current camera (green); **(b)** covisibility graph, with all the edges connecting keyframes; **(c)** Essential graph, only the edges with high weight are present and a loop closing edge in red. Source:[35]

Chapter 3

Implementation

Contents

3.1 Dataset

The dataset was created from video footage captured during ROV Luso missions, in 2009, exploring hydrothermal vents along the archipelago of Azores. Since the Azores have abundant vulcanic activity, it hosts some of the largest hydrothermal vents fields worldwide, like Menez Gwen and Lucky Strike. From Menez Gwen, 9 videos were collected, amounting to nearly 3 hours of video footage. As per Lucky Strike, 20 videos were gathered, totalling nearly 12 hours of video footage, which in total gave us 15 hours of video data. At the end, we were able to create a dataset with 1200 pairs of images and semantic segmentation ground truths.

The process of gathering samples consisted of three steps: First, going through all 15 hours of video footage and selecting the periods of each video that contained images suited for the dataset; Then, creating a script that from each segment of the video deemed valid, would extract 1 out of each 5 frames. And since the video was recorded at around 25 frames per second, we are extracting 5 frames out of each second of video. Finally, from all the images yielded by this process, clear those that are not well suited for the purpose of this work. For instance, images from the ROVs descent and arise,

since the only surroundings are water and there are no foreground instances. Images with excessive noise, like blur, but also where only dust and bubbles are present were discarded. The goal was to keep images where a foreground and background can be identified. The foreground consisting of rock and seabed, that clearly pose as obstacles, and the background composed by ocean or rock/seabed of different characteristics. Having in mind that we want to develop a robust model, capable of providing good generalization, without compromising when too much noise is present, the process of choosing good candidates for samples included searching for images containing suspended particles, bubbles, blur and other types of disturbances.

The methodology followed to create this dataset, after having all the inputs gathered, followed three base steps: applying contrast enhancement; produce an estimate segmentation mask, using edge detection tools; applying the necessary corrections, to yield ground truths as accurate as possible (Fig. 3.1).

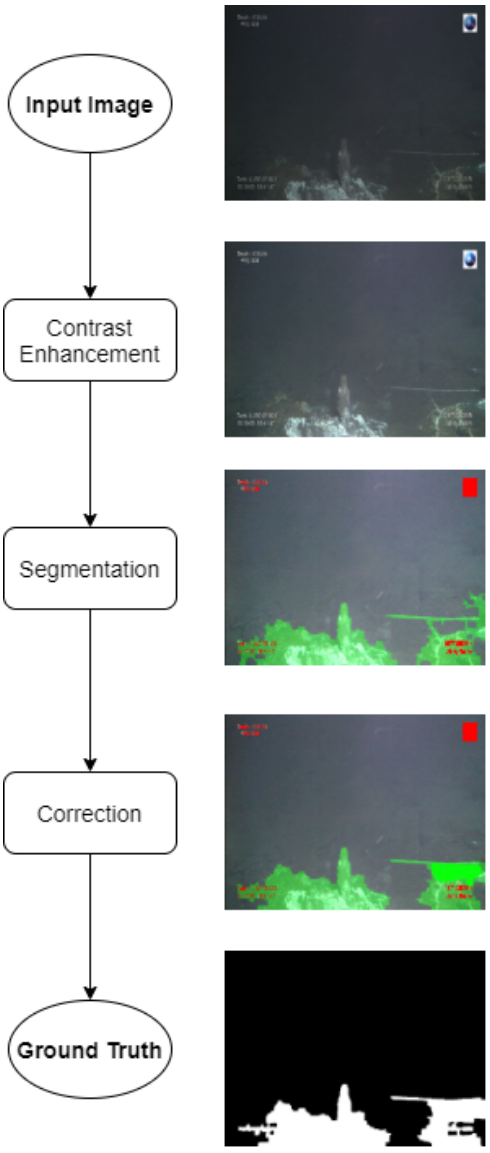
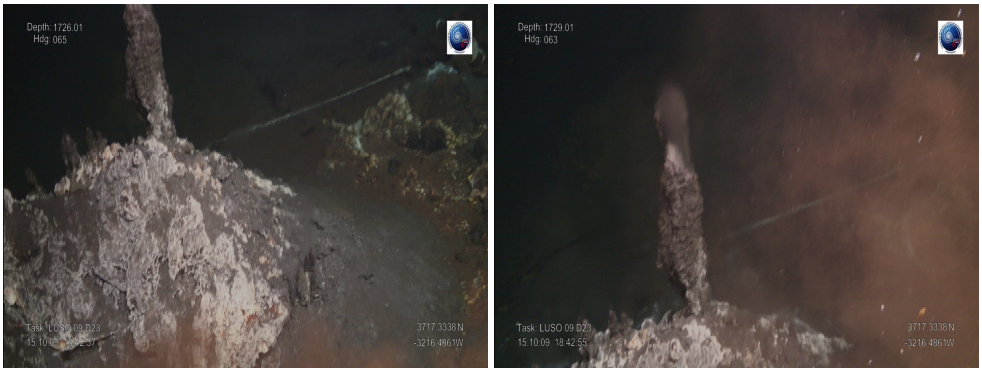


Figure 3.1: Dataset creation flow chart.

3.1.1 Contrast Enhancement

Extracting features of images with very low lighting and uneven contrast can be extremely difficult. In underwater environments, light suffers from a great amount of scattering. Image channels, such as red, are severely attenuated, resulting in images with low contrast and with dominant green and blue channels, considering an RGB colour system.

In underwater environments, light suffers severe scattering effects, making the captured images look hazy or foggy. Moreover, in underwater medium, conditions change in short time frames, caused by changes in tide, floating particles or regions of high turbidity (Fig. 3.2). In the context of autonomous navigation, this problem becomes particularly outstanding. Since deep sea environments are denied of Global Positioning Systems (GPS), visual information is the main source of information and guidance. This creates the need for processing tools that keep our data clean, with balanced contrasts, minimized colour distortion and good lighting conditions.



(a) (b)

Figure 3.2: Images of the same region, captured with few seconds of difference

In recent times, many visibility enhancement methods have been proposed. Among them, He et al. [36] proposed a solution which is widely regarded as a simple, but accurate and efficient solution to his kind of problem, which states that in low contrast scenes, some pixels often have low intensity on at least one colour channel. This is known as Dark Channel Prior (DCP). Several other methods have been proposed, but comparative results [37] show that DCP is still an excellent approach to choose. A joint study between the University of Victoria (BC, Canada) and Ocean Networks Canada [38], which operates cabled ocean observatories, using cameras to collect data of seabed in deep depths where only artificial lighting is present, states [36] as an efficient approach to their problem.

When light travels in turbid mediums, like water or atmosphere, it suffers from absorption and scattering, which result in degraded images, with low contrast and poor colour quality. This scattering is not homogeneous across the scene, since this effect depends on the distance of scene points to the camera. Furthermore, in the conditions that this footage was produced, where natural lighting is non-existent, properly illuminating the entire captured scene is impossible, resulting in dark areas that can compromise the data analysis.

The Single Scattering Atmospheric Model (SSAM), models a hazy image, **I(x)**, as

$$I(x) = J(x)t(x) + A(1 - t(x)), \tag{3.1}$$

where **x** is a coordinate vector of a given pixel, **J** is the haze-less image, **A** is the atmospheric light and **t** is the light that reaches the camera without scattering. If we calculate **t** and **A**, given the input **I** we can recover **J** and have the enhanced haze free image.

The first thing to be done is calculate the *dark channel*. The dark channel is formed by the pixels with the lowest intensity in one of the three RGB channels, in a patch of given size. The dark channel, $\mathbf{J^{dark}}$, is given by

$$J^{dark}(x) = \min_{y \in \Omega}(\min_{c \in r,g,b} J^c(y)) \tag{3.2}$$

where $\mathbf{\Omega}$ is a patch centered at **x**.

Considering what we first introduced about the DCP, we can say that if **J** is a haze-free image, with recovered contrasts, then

$$J^{Dark} \to 0. \tag{3.3}$$

The atmospheric light **A**, or the scene light, is calculated by choosing the 0.1% brightest pixels of the dark channel. This corresponds to the area with the most haze and lower contrasts, as the scattering is highly dependent on the distance travelled by light. The same pixels are retrieved from the original image **I**. Then, the mean of this group of pixels is calculated for each RGB channel.

Another necessary step for the improvement of image quality is to determine the transmission map, from the hazy image equation (eq. 3.1), which can be written as

$$t(x) = 1 - w \min_{y \in \Omega(x)} \left(\min_{c} \frac{I^c(y)}{A^c} \right) \tag{3.4}$$

where $\mathbf{A^c}$ is the atmospheric light in each color channel, considering pixel intensities ranging from 0 to 1. Our understanding of human vision considers that haze allows the perception of distance and depth, an occurrence denominated aerial perspective.

If the transmission map was to be used just like it is yielded from equation 3.4, we would obtain an output image with undesired artifacts (halos or pixelated blocks) around objects present in the scene. In order to prevent this, the transmission map must be refined (filtered). The approach proposed by Tunai et al. [38] consists of using a guided filter. According to which, a filtered image $\mathbf{q}$ can be recovered from a guidance image $\mathbf{I}$ and an input image $\mathbf{p}$ using

$$q_i = a_k I_i + b_k, \forall i \in w_k \tag{3.5}$$

with $\mathbf{i}$ being the pixel's index and and k the index of a local square window $\mathbf{w}$ of radius $\mathbf{r}$. The values of

$\mathbf{a_k}$ and $\mathbf{b_k}$ can be determined using:

$$a_k = \frac{\frac{1}{|w|}\sum_{i \in w_k} I_i p_i - \mu_k \overline{p}_k}{\sigma_k^2 + \epsilon} \tag{3.6}$$

$$b_k = \overline{p}_k - a_k \mu_k \tag{3.7}$$

where $\mu_\mathbf{k}$ and $\sigma_\mathbf{k}^\mathbf{2}$ are the mean and the covariance of $\mathbf{I}$ in $\mathbf{w_k}$, $|\mathbf{w}|$ is the number of pixels in $\mathbf{w_k}$ and $\mathbf{p_k}$ is the mean of $\mathbf{p}$ in $\mathbf{w_k}$. The equations above were provided by He et. al at [39].

After determining the scene light $\mathbf{A}$ and the filtered transmission map $\mathbf{t}(\mathbf{x})$ we have the necessary information to recover the the enhanced image $\mathbf{J}(\mathbf{x})$, from equation 3.1 comes

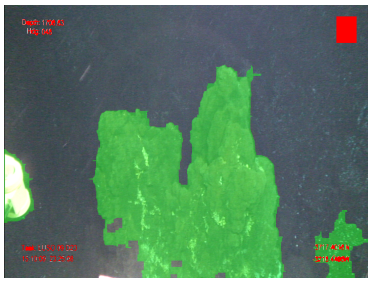
$$J(x) = \frac{I(x) - A}{\max(t(x), t_0)} + A \tag{3.8}$$

where $\mathbf{t_0}$ is a constant introduced to limit the value of the denominator. For certain denominator values, the recovered image can be prone to noise in the most hazy regions, so it is good practice to introduce a constant ($\mathbf{t0}$) and limit the denominator value.

The parameter values chosen for this implementation were $w = 0.5$ (Eq. 3.4), $t_0 = 0.6$ (eq. 3.8), using patches of dimensions 15×15.

In figure 3.3 we can see the difference this contrast enhancement technique does in the predicted segmentation mask. On a series of images, this results in a lot of time saved, as correcting masks becomes much faster.

(a) Raw image (b) Enhanced image (c) Mask prediction for raw image



(d) Mask prediction for enhanced image

Figure 3.3: Output comparison for raw and enhanced images.

3.1.2 Operator Selection

Annotating an entire dataset by hand would be extremely time consuming. Thus, developing a methodology to help create the dataset is advantageous in what concerns both efficiency, since the number of pixels being annotated by hand is greatly reduced, and accuracy, considering that having specific and objective criteria to classify pixels reduces human classification errors. The last was a big concern, as in an image with over one million pixels, it is impossible for human eye to distinguish each pixel with clarity in a neighbourhood of an edge or border. So, developing an approach that yields the most correct ground truth possible was key for the progress of this work.

Analysing the dataset, a characteristic becomes evident: most of the images are very rich in edges and corners, due to the presence of shellfish and algae on the seabed. Therefore, edge detection operators were chosen to extract those features from the images. Besides the edge detection approach, the k-means clustering algorithm for segmentation was also tested.

Sobel, Canny, Prewitt and Laplace

The edge detection methods tested were based on Sobel, Prewitt, Laplacian and Canny filters. The process is very similar for the four cases. There is a filter (kernel) that is convoluted accross the image. These kernels are an estimate of the derivative of pixel intensity, estimating the direction of the highest variation of pixel intensity (from bright to dark), which provides insights on how the the pixel intensities changes in each given point, and allows intensity gradient estimation. In regions where intensity variation is significant, we can say that it is a separation between objects and, thus, we are in the presence of an edge. All these filters perform 2-D spatial gradient measurement, which means that a gradient is estimated across the horizontal (G_x)and vertical (G_y) dimensions and are then used to calculate the absolute gradient magnitude

$$G = \sqrt{G_x^2 + G_y^2}. \tag{3.9}$$

In theory, Sobel operator typically uses 3×3 kernels, however for improved segmentation results 5×5 kernels are often used [40]. These kernels usually take the following values

$$G_x = \begin{bmatrix} +2 & +2 & +4 & +2 & +2 \\ +1 & +1 & +2 & +1 & +1 \\ 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & -2 & -1 & -1 \\ -2 & -2 & -4 & -2 & -2 \end{bmatrix} G_y = \begin{bmatrix} +2 & +1 & 0 & -1 & -2 \\ +2 & +1 & 0 & -1 & -2 \\ +4 & +2 & 0 & -2 & -4 \\ +2 & +1 & 0 & -1 & -2 \\ +2 & +1 & 0 & -1 & -2 \end{bmatrix}. \tag{3.10}$$

In the Prewitt operator, used to estimate the first derivative of the intensity, the literature suggests that 3×3 kernels are to be used and they take the following values

$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +1 & 0 & -1 \\ +1 & 0 & -1 \end{bmatrix} G_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ +1 & +1 & +1 \end{bmatrix} \tag{3.11}$$

While Sobel and Prewitt kernels use a first order derivative mask, the Laplacian operator uses a second order derivative masks. This operator usually comes in two forms: a positive operator and a negative one. The positive Laplacian operator is used to retrieve the outer edges of an image and its values are

$$L = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}. \tag{3.12}$$

The negative Laplacian operator is useful in extracting inner edges

$$L = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}. \tag{3.13}$$

The implementation of this method was done using the OpenCV library for Python, and their default operator for the Laplacian algorithm is the positive one[1] (equation 3.12).

The Canny edge detection method was also implemented through the OpenCV library, which has a specific method for this operator [2]. The Canny algorithm was proposed by John F. Canny in 1986 in a paper that followed is PhD. thesis [41]. The algorithm follows several stages:

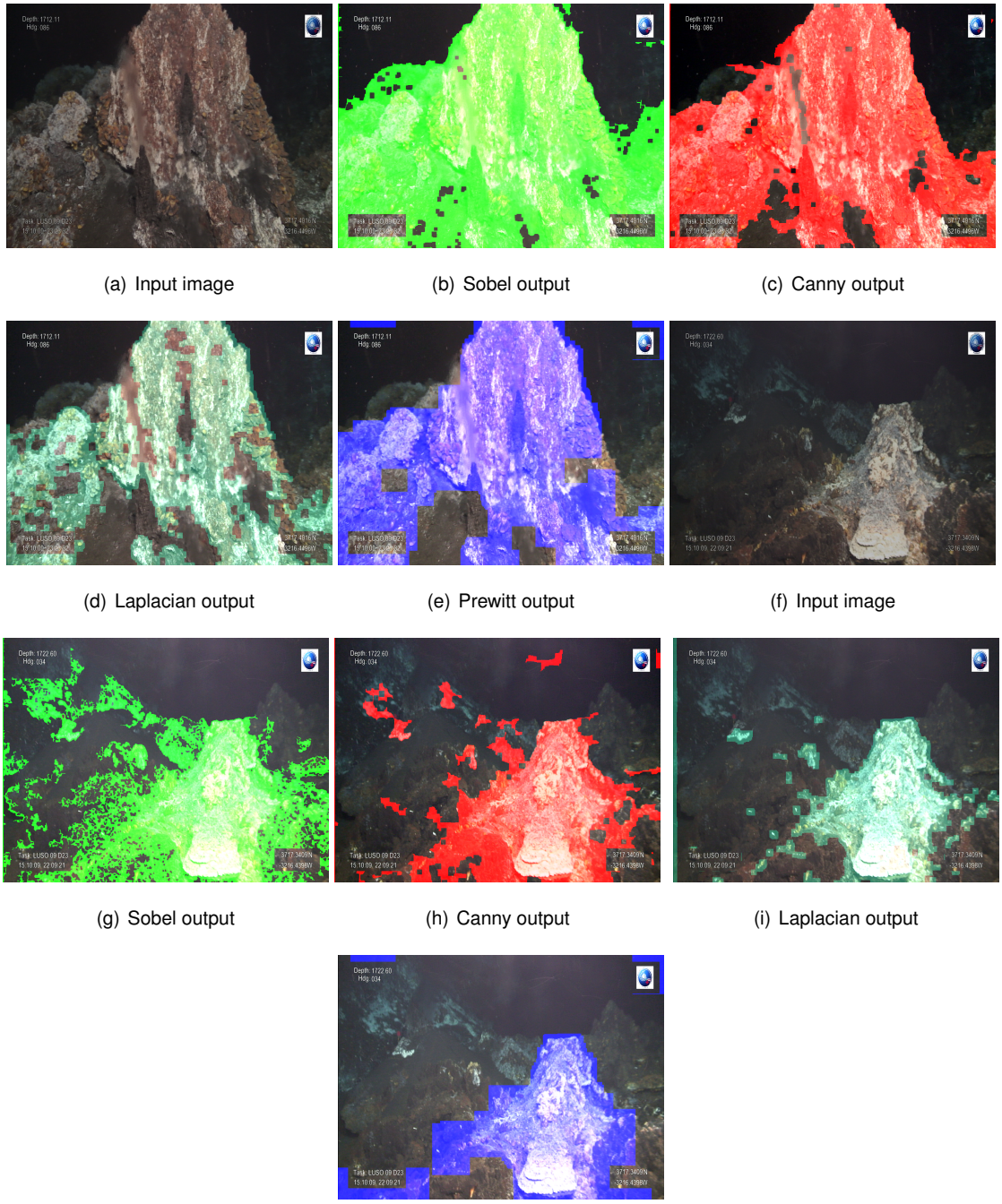
1. Noise reduction, using a 5×5 Gaussian filter;
2. Finding the gradient of pixel intensity in vertical and horizontal directions. The library used for the implementation computes the gradients through Sobel filters;
3. Non maximum suppression, which consists in removing pixels that are not edges. This is done by checking for each pixel if it is a local maximum in its vicinity, in the direction of the gradient;
4. Hysteresis Thresholding: at this stage, all the edges are reviewed to conclude which ones are edges and those who are not. To do this, two thresholds must be specified, a minimum and a maximum. The edges with intensity above the maximum threshold are sure to be edges and the ones with intensity bellow are discarded. The edges with values between the two thresholds are considered edges if they are connected to sure edges and discarded if they are not.

The output of this method is an edge map similar to those of the previously described operators. The method in the OpenCV library takes 4 arguments: the input image, the lower and upper threshold values for the hysteresis thesholding, and the kernel size for the gradient estimation operator. After several tests, the configuration of parameters that yielded the best results was a minimum threshold of 0, an upper threshold of 58 and a 3×3 kernel.

To prevent the detection of edges that appear in the image due to noise, the images must be filtered. The median filter from OpenCV [3] was used with kernels of size $ksize = 5$ for the four operators (Sobel, Prewitt, Laplacian, Canny). After filtering, the edge maps are retrieved by convoluting the kernels across

[1] https://docs.opencv.org/master/d5/d0f/tutorial_py_gradients.html
[2] https://docs.opencv.org/master/da/d22/tutorial_py_canny.html
[3] https://docs.opencv.org/3.4/d4/d86/group__imgproc__filter.html#ga564869aa33e58769b4469101aac458f9

the image. The edges alone are not a segmentation mask, because pixels are not labeled individually, instead only the pixels that are part of edges are labeled. However, if we perform morphological operations, such as closing and dilation [4], we are able to create a mask that, if overlayed in the original image, covers the entire object instead of just the edges. To remove unwanted areas, morphological transformations such as opening and eroding can be implemented. Furthermore, a function was developed to erase areas that are bellow a certain size threshold, since most objects that would be segmented are large and unwanted areas of dust and other particles are small in comparison.



(a) Input image (b) Sobel output (c) Canny output

(d) Laplacian output (e) Prewitt output (f) Input image

(g) Sobel output (h) Canny output (i) Laplacian output

(j) Prewitt output

Figure 3.4: Comparison between edge detection operators for segmentation.

[4] https://docs.opencv.org/master/d9/d61/tutorial_py_morphological_ops.html

Analysing figure 3.4 we can state that the method that yields the best results is the one based on the Sobel operator. Not only does it classify more pixels correctly, it also avoids segmenting areas that are not seabed, such as noise or areas with more illumination as is the case in images **(g)** and **(h)** of figure 3.4.

K-means Clustering

In order to evaluate different approaches, besides the edge detection methodology, a machine learning based algorithm, K-means clustering, was tested.

K-means clustering is an unsupervised learning algorithm (meaning that when a model is trained, there is no labeled data) that was designed for signal processing, aiming to partition data into k clusters, where each instance of data is assigned to nearest cluster centroid. For an image with width W and height H, the iterative process of this algorithm can be described as [42]:

1. Assign k random pixels as the initial centroids;

2. Calculate the distance of each pixel $p(w, h)$ to all the centroids c_k

$$d = ||p(w, h) - c_k|| \tag{3.14}$$

3. Based on the distance, each pixel is assigned to nearest cluster with the nearest centroid.

4. The centroids for each of the k clusters are computed again

$$c_k = \frac{1}{N} \sum_{i}^{N} p(w, h)_i \tag{3.15}$$

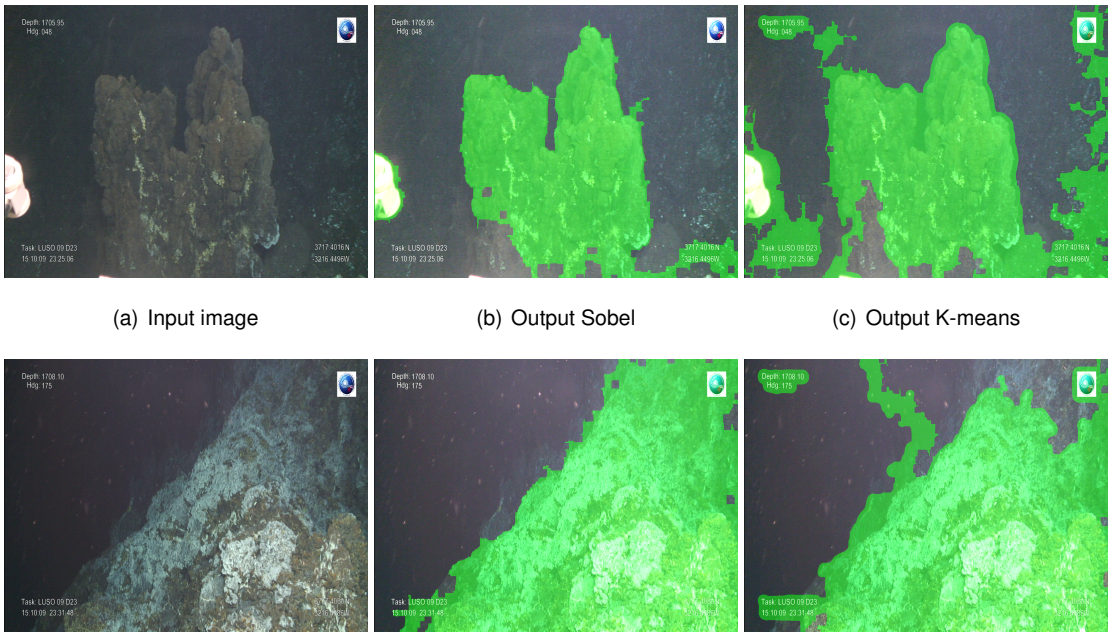
   where $\mathbf{N}$ is the number of pixels in the $\mathbf{k}$ centroid.

5. Repeat the process until the maximum number of iterations or the specified accuracy (tolerance) is reached.

Again, the implementation of this method was done via OpenCV library, which provides a trained k-means clustering model [5]. The steps leading up and following the algorithm are very similar to what was done for the edge detection algorithms. Meaning that first the image is filtered to clear the presence of excessive noise and after the k-means method a series of morphological transformations are performed to assign as many pixels as possible with the correct label.

A number of 10 clusters was specified in the parameters of the k-means method. Also, for computational efficiency reasons, the maximum number of iterations was set to 10 and an accuracy of $\epsilon = 1.0$.

[5] `https://docs.opencv.org/master/d1/d5c/tutorial_py_kmeans_opencv.html`



(a) Input image (b) Output Sobel (c) Output K-means

(d) Input image (e) Output Sobel (f) Output K-means

Figure 3.5: Comparison between the Sobel and K-means clustering algorithms.

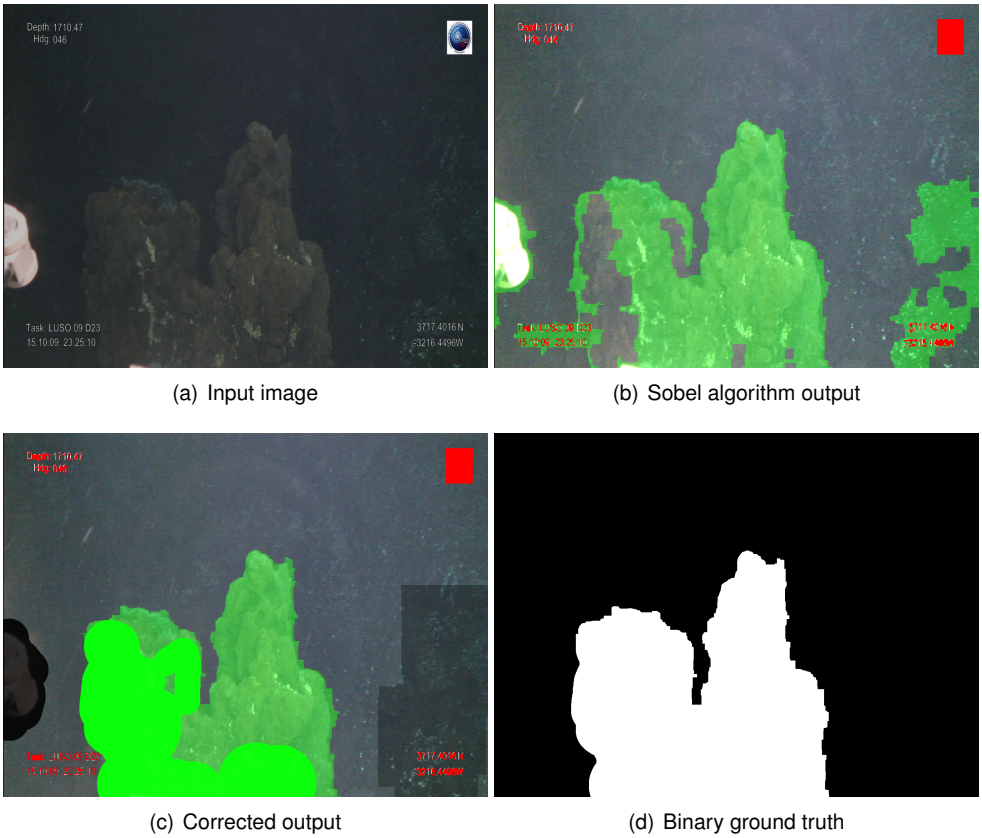
In figure 3.5 we can see a qualitative comparison between the k-means algorithm and the Sobel operator, which was the best performing edge detection method. Again, we can see the Sobel solution performing better. The K-means is prone to identifying particles and noise areas as foreground, even after being submitted to noise filtering, as we can state from comparing images **(b)** and **(c)**. Furthermore, Sobel is also better at correctly classifying more pixels, if we take a closer look at images **(e)** and **(f)**. This differences in performance are consistent throughout the dataset, with Sobel being the most robust solution. Over a set of many hundreds or thousands of images, the time spent correcting the solutions of both algorithms will be substantially different, with Sobel being the most efficient one, since there are less areas to amend. Therefore, the chosen method to create a first iteration of the ground truth creation process was the Sobel solution.

3.1.3 Ground Truth Creation

The methodology presented in section 3.1.2 is only the first step for creating a dataset of paired input images and segmentation ground truths. Since the segmentation tool based on Sobel operator algorithm is not robust enough to generate accurate ground truths for the entire dataset, a tool for correcting segmentation masks by hand was developed. This tool consisted of two brushes, one to colour in green the areas that should be segmented and a another to cover in black areas of increased lighting, and one eraser, used to delete areas that were mistakenly identified as foreground. It is possible to increase and decrease the size of the brush by scrolling, to make the colouring process faster in inner parts of foreground objects and improve accuracy around edges, respectively. After all the adjusts were made, each mask was converted to a binary format, since this is the type of ground truth input accepted by the chosen deep learning model.

The images contain alphanumeric characters on the top and bottom corners, which shall not be identified by the segmentation algorithm as foreground, therefore this should be taken in consideration when creating the ground truths. These characters are static, meaning that they are always on the same coordinates. Furthermore, they are always white and brighter than the background. So, for these regions of the image, a threshold based segmentation was implemented. With the masked characters, a red color is overlayed on the characters for better visualization in the correction phase. When converting to the binary ground truth, these areas are placed in black, because the colour label is different than that of the foreground.

The conversion of the coloured masks to binary ground truths is done by identifying which areas have high pixel intensity on the green channel. The black brush is particularly useful for this task, since areas with high illumination or whitish colouration have high pixel intensities in all three channels (RGB images) and would be considered foreground in binary format if not covered in black to lower pixel intensity in every channel.



(a) Input image (b) Sobel algorithm output

(c) Corrected output (d) Binary ground truth

Figure 3.6: Ground truth creation stages. The green colour identifies foreground areas and red signals areas to be discarded, such as the characters and the logo. On the binary ground truth, in white we have the foreground instance and black the background.

3.2 Fully Convolutional Network

The chosen framework for the implementation of the deep learning solution for semantic segmentation was Pytorch, developed by Facebook AI Research Lab and is seeing its usage increase at good pace. Comparing to other frameworks, such as Tensorflow or Keras, Pytorch is very flexible, offers good debugging capabilities and runs faster, meaning shorter training duration.

3.2.1 Data Loading and Preprocessing

The first step of the data loading process consists of cropping the images. The implemented model only accepts input images whose dimensions are multiples of 16 since we have 1440 pixels of width, which already is a multiple of 16, the only dimension that should be cropped is the height. The original image's height is 1080 and it should be converted to 1056.

Also, images should be converted from RGB format to BGR and pixel intensities for the three channels are normalized, so that they stay in the range $[-1; 1]$ instead of $[0; 255]$.

3.2.2 Architecture

The research by Long et al. [12] evaluates the performance of three different fully convolutional networks (FCNs), the FCN32s, FCN16s and FCN8s. The one that achieved the best results in [12] was the FCN8s, since it combines the output of deeper layers with outputs of shallower ones, preventing the loss of spatial information that is crucial to capture all the details. This operation of combining results of different layers will be further explained ahead.

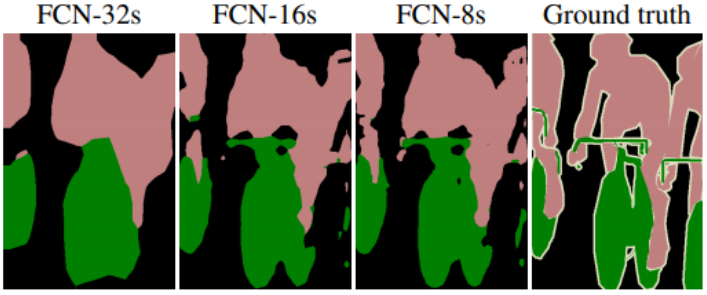


Figure 3.7: Comparison between the results achieved by the FCN32s, FCN16s and FCN8s. Source:[12]

The implementation of the FCN8s can be split in two stages. First, a transfer learning approach was used where a pretrained VGG16 network imported from Pytorch is integrated for the convolutional part of the network. The traditional VGG16 consists of 5 convolutional and pooling layers followed by fully connected layers. However, since we want the network to accept inputs of arbitrary size, the fully connected layers are replaced with another two convolutional layers, performing the convolutions with kernels of size 1×1. Second, the transpose convolution (deconvolution) layers are added at the output of VGG16 to upsample results to the size of the input image, providing an image where each pixel is assigned to a class. Figure 3.8 illustrates a comparison between the layers of a CNN and a FCN.

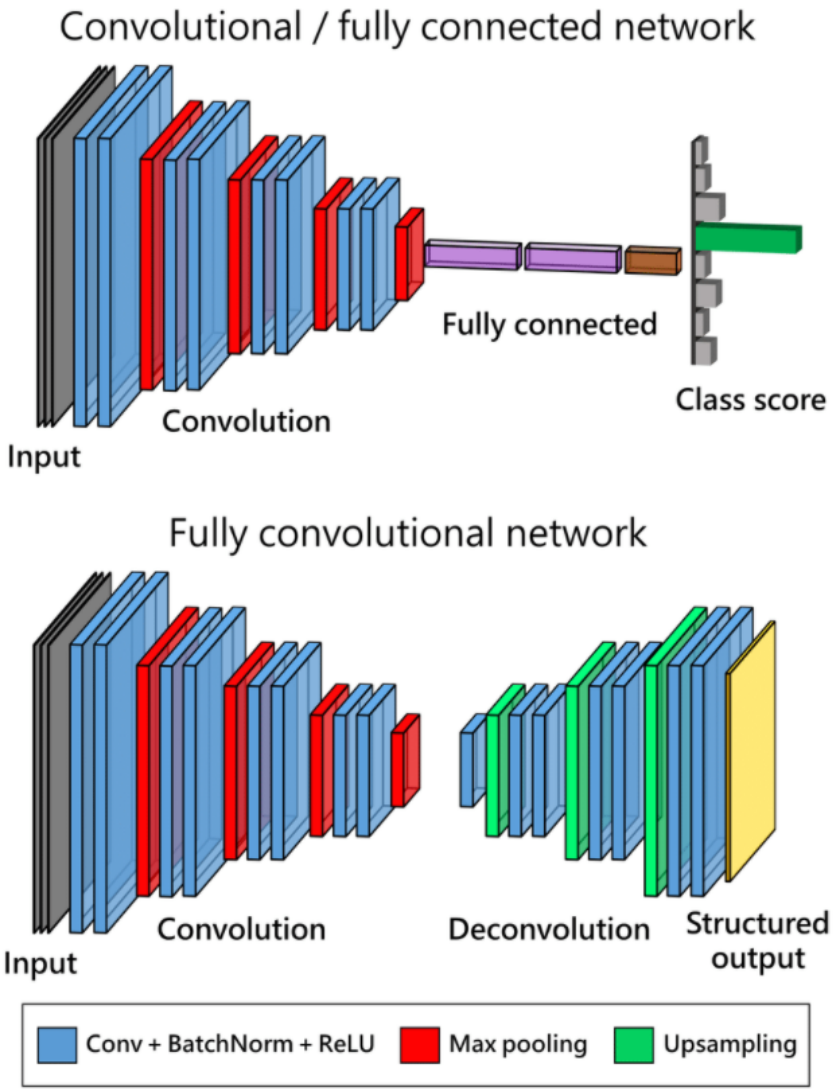


Figure 3.8: Comparison between the layers of a Convolutional Neural Network and a Fully Convolutional Network. The CNN and the FCN are coupled, by removing the fully connected layers of the CNN and adding deconvolution layers to upsample the pixelwise class predictions. Source: [43]

This architecture yields good results because it combines results from different layers, making local pixel predictions while respecting global structure. The outputs of the 3^{rd}, 4^{th} and 5^{th} convolutional layers from VGG16 are combined with results from the transpose convolution layers(figure 3.9). When we go deeper in the network, we lose spatial information, due to all the convolution and pooling layers. So, if we combine the output of deeper layers with the output of shallower ones, we are adding location information, it is expected that the quality of the results increases, due to increased detail. This combination is an elementwise sum. The output of the 5^{th} convolutional layer is upsampled by a factor of 2 and summed with the output of the 4^{th} convoluional layer. The output of this operation is again upsampled by a factor of 2 and summed with the output of the 3^{rd} convolutional layer, which is later upsampled by a factor of 8 to yield the final result.

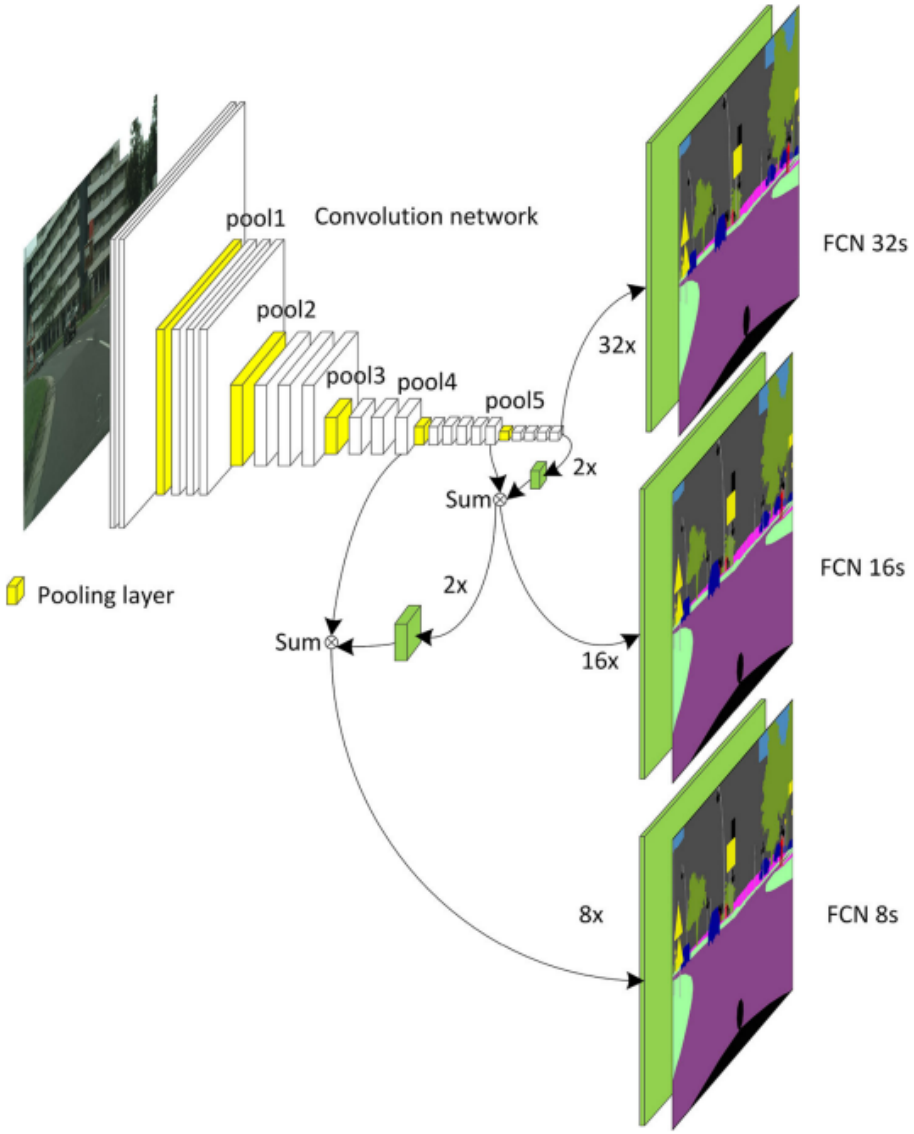


Figure 3.9: Graphic representation of the combination of results from different layers. This process is used to improve the quality of results, by adding feature maps from deeper layers. Source: [44]

The chosen activation function used at the end of each layer was the rectified linear unit (ReLU), as it is the most commonly used for CNNs and FCNs. The selected loss function was binary cross entropy (BCE) and the optimization process is done through stochastic gradient descent, using the RMSProp optimization algorithm.

3.2.3 Network Parameters

Regarding the parameters chosen for the training phase, for computational capacity reasons the batch size was set to 1, as images have a very large number of pixels and consume a great ammount of memory, even training on GPUs using the CUDA toobox from Nvidia. Learning rate was set to $1 \cdot 10{-}5$, weight decay $1 \cdot 10^{-5}$, step size 50 and momentum 0. The training dataset was iterated across 130 epochs.

To arrive to this configuration of parameters several tests were executed, by training the model with

different parameters. The values chosen for the first training session were the ones provided by the paper of Long et. al [12] and different sets of parameters were experimented from there.

3.2.4 Dataset Management

The dataset consists of 1198 images, of which 798 were used for training and 150 for validation throughout each training epoch. After training, the model was tested with a dataset of 150 images, with examples that were not used in the training phase. Furthermore, since the model was trained several times with different parameters, a dataset with 100 unique examples from a different underwater site was stored to evaluate the model performance after the configuration of training parameters was settled. The goal of the last is to assess how robust the model is. No data augmentation was performed, since the work of [12] states that performing such task for does not perceptibly improve network performance and increases the time required to train.

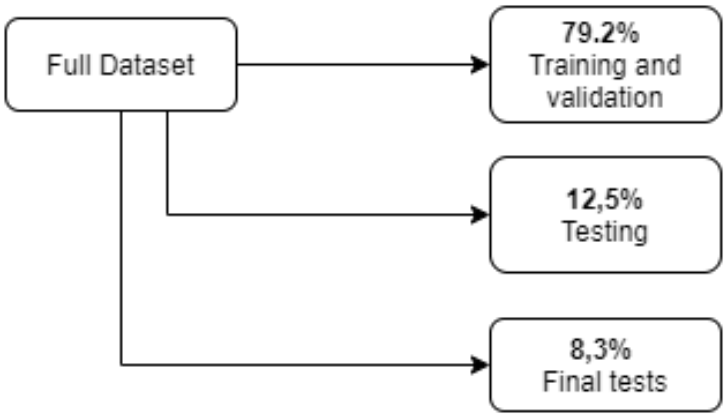


Figure 3.10: Schematic of the dataset management. A dataset with images from a different underwater site was stored to evaluate the model after all the network hyperparameters were settled.

3.2.5 Evaluation Metrics

To evaluate the network's performance during training and testing sessions the selected performance metrics were mean pixel accuracy and intersections over union, as these are the ones considered most relevant in the work of Long et al. [12].

The mean pixel accuracy (MPA) can be defined as

$$MPA = \frac{CP}{TP} \tag{3.16}$$

where CP is the number of correct pixel classifications and TP is the total number of predictions. The number of correct pixels is, in fact, a sum of the number of true positives and true negatives. Thus, this metric can be deceiving in a binary problem if the class representation is scarce, as it will essentially tell how well the true negative cases are identified. So, the need for a different metric to corroborate results is legitimate.

The intersections over union (IoU), sometimes also called Jaccard index, quantifies the overlap between the ground truth mask and our prediction mask. In other words, it measures the number of pixels common between the target and the prediction and divides by the total number of pixels present on both masks. [45]

$$IoU = \frac{target \cap prediction}{target \cup prediction} \tag{3.17}$$

The intersections over union are determined for each class separately. Then the average of the IoUs is determined to provide a global metric of the segmentation predictions.

Chapter 4

Results

Contents

An analysis of the results achieved by the segmentation neural network (FCN) is done in this chapter, by evaluating a series of experiments. Furthermore, the impact of the achieved segmentation masks is evaluated when SLAM algorithms are applied, to assess if there is an improvement in performance by applying the proposed methodology.

4.1 Fully Convolutional Network

This section's objective is to evaluate the results attained by the Fully Convolutional Network. The model was trained a total of 8 times, with different configurations of hyperparameters and correction of the dataset between each training phase. After determining the configuration of parameters that achieves the most accurate results, the model is tested on a set of images, collected from a different site. The results of these last test are also exposed in this section.

4.1.1 Training an FCN with Different Parameters

These interim tests are the set of tests that aim at reaching the optimal configuration of network training parameters for this problem. All the tests were performed on the same dataset, as seen on figure 4.1. After each epoch of training, a validation set is ran through the network to evaluate performance

evolution throughout the epochs of training. For each training phase, the validation dataset is always a random subset of 150 images from the overall training set.

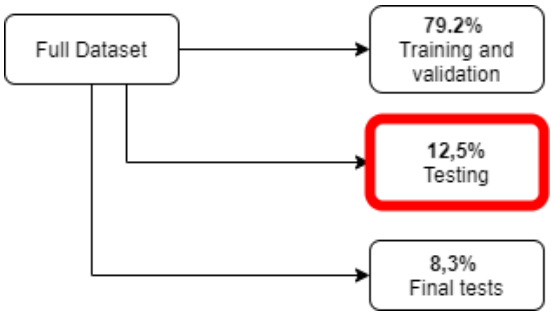


Figure 4.1: Dataset management chart. The red box denotes the part of the dataset used for testing throughout the tests exposed in this section. The training and validation sets are the same throughout all tests. The last one will be discussed on section 4.1.2.

For the first training of the neural network, the training parameters were as of table 4.1, which were extracted directly from the paper of Long et. al [12]. The number of epochs was decreased from that of the paper, due to the smaller dataset of this thesis in comparison to the one used in [12]. Also, the batch size was set to one, due to memory usage constraints.

This first test did not follow the dataset structure of figure 4.1, as the purpose was to evaluate if the FCN was a good model to train and tune for this problem. For this test 898 images were used for training, 100 for validation and 100 were used for testing.

Test	Epochs	Batch Size	Learning Rate	Weight Decay	Step Size	Gamma	Momentum
1	100	1	1×10^{-4}	1×10^{-5}	50	0.5	0
2	100	1	1×10^{-4}	1×10^{-5}	50	0.5	0
3	100	1	1×10^{-4}	1×10^{-5}	50	0.5	0
4	130	1	1×10^{-4}	1×10^{-5}	50	0.5	0
5	100	1	5×10^{-5}	1×10^{-5}	50	0.5	0
6	130	1	5×10^{-5}	1×10^{-4}	50	0.5	0
7	130	1	5×10^{-5}	1×10^{-5}	50	0.5	0

Table 4.1: Network parameters for each test.

Figure 4.2 (**a)** and **b)**) shows the evolution of mean pixel accuracy and mean intersections over union (IoU) as the number of epochs increases. We can see it converge at around 70 epochs, but stabilization only occurs after more than 80 epochs. In tables 4.2 and 4.3 we can see the achieved results on these metrics for the validation and test set, respectively. Since the network achieved high pixel accuracy and IoU, the FCN was trained and tuned on the next tests.

Between the first test and the second, no networks parameters were changed (table 4.1). However, the training, validation and testing sets were changed to match the proportions shown on figure 4.1. In this phase, the model was trained with 798 images, validated on 150 images and tested on 150 images. The dataset distribution for this test is the one that will be used for all the following tests. Analysing figure 4.2 (**c)** and **d)**) and tables 4.2 and 4.3 there were no significant changes if compared to the previous test.

Again, for the test number 3, no network parameters were changed (table 4.1). However, there were some segmentation ground truths that had some minor misclassified areas, which were corrected

Test	Pixel Accuracy	Mean IU
1	96.8%	92.79%
2	96.7%	92.21%
3	96.97%	93.13%
4	96.62%	92.23%
5	97.17%	93.43%
6	96.80%	92.48%
7	96.70%	92.53%

Table 4.2: Results achieved on the validation set for each configuration of parameters.

Test	Pixel Accuracy	Mean IU
1	98.1%	96.03%
2	97.76%	94.37%
3	97.97%	94.71%
4	98.03%	94.82%
5	98.09%	95.09%
6	98.07%	94.84%
7	97.9%	94.61%

Table 4.3: Results achieved on the test set for each configuration of parameters.

before starting the training phase for this test. Looking at figure 4.2(**e)** and **f)**) and tables 4.2 and 4.3, the conclusion is that no major changes happened.

For the training and testing phase number 4 the only parameter that changed was the number of epochs, which was increased to 130 (table 4.1), to evaluate if increasing the number of epochs would make the model converge to a more optimal solution.

The impact of this new configuration of parameters was not meaningful, with only a slight improve in metrics of the test set, see tables 4.2 and 4.3 for test 4, although the graphics of figure 4.2(**g)** and **h)**) appear more stable at the end of the training phase than the previous ones. Thus, it can be concluded that 100 epochs is enough to train the model to achieve good accuracy, if the rest of the parameters are kept with the same values as the other tests. Furthermore, increasing the number of epochs above a number that yields a good solution may contribute to overfitting, which translates to model too close to the training data but that cannot generalize well for different data samples.

When choosing the 5^{th} configuration of parameters, the learning rate was decreased from 1×10^{-4} to 5×10^{-5} and the number of epochs decreased from 130 to 100, in comparison to the previous test. The decrease in the number of epochs was due to not seeing significant impact in the previous test, where it was increased. Decreasing the learning rate decreases the rate at which the weights are updated, thus making convergence slower but possibly leading to a more optimal solution.

Taking a closer look at tables 4.2 and 4.3, one can say that the impact of these changes was not meaningful, since there was no considerable change in both performance metrics, when compared to the previous experiments. Furthermore, the graphics of figure 4.2 (**i)** and **j)**) show a 'spike', which may indicate that convergence was not yet reached.

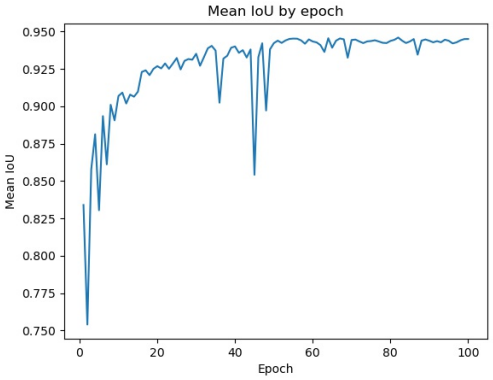
On the following test (number 6), the weight decay was increased to 1×10^{-4} and the number of epochs was set to 130, to evaluate if more epochs would lead to a more optimal solution after changing the learning rate in the previous experiment. Lower values of weight decay leads to models that take longer to fit, thus increasing the weight decay will theoretically lead to faster convergence. Furthermore, since with lower values the model takes longer to fit, it may also lead to overfitting.

Although changes were made, the model still fitted very closely to what was seen in previous experiments, as can be stated after analyzing tables 4.2 and 4.3, and the graphics of figure 4.2 (**k)** and **l)**).

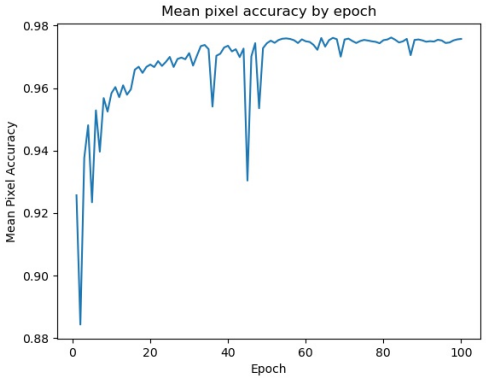
From the previous experiment one can say that having an increased weight decay was not advantageous for this specific model and dataset. Therefore, for the 7^{th} training and testing period, it was reset to the initial value of 1×10^{-5} and all the other parameters were kept identical to the past two experiments.

Looking at results of tables 4.2 and 4.3 for the test number 7 and comparing to the previous iterations, the performance difference is not significant. However, the graphic of figure 4.2 (**m)** and **n)**) shows that the model is very stable after 130 epochs.

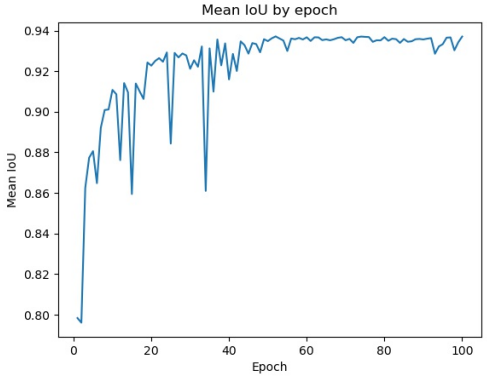
On the test set used throughout the 7 experiments the methodology based on the Sobel operator that was explained on chapter 3 achieved an average pixel accuracy of 91.8%. The difference may not seem drastic, if compared to nearly 98% of the FCN solution, but the FCN is more robust when it comes to properly classifying alpha-numerical characters present in the footage. Furthermore, the FCN is not prone to identify floating particles or ROV parts as foreground, contrary to the Sobel-based solution.



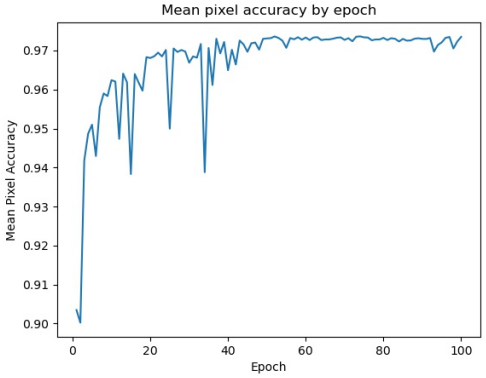
(a) Mean IoU evolution for 1^{st} test.



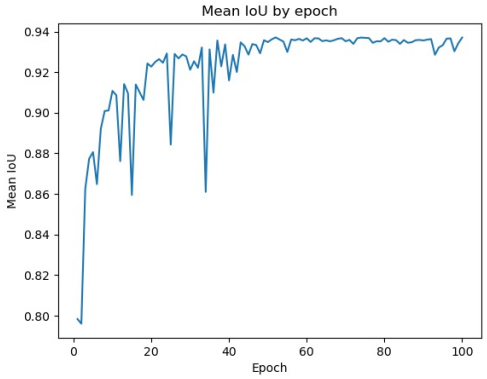
(b) Mean pixel accuracy evolution for the 1^{st} test.



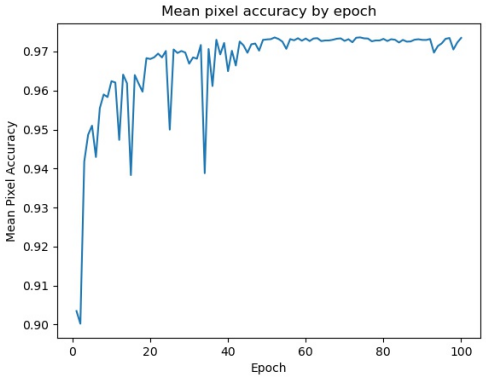
(c) Mean IoU evolution for 2^{nd} test.



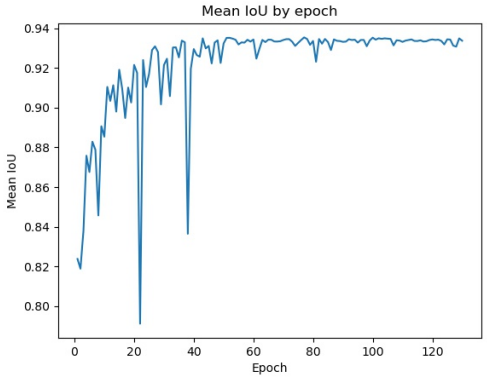
(d) Mean pixel accuracy evolution for 2^{nd} test.



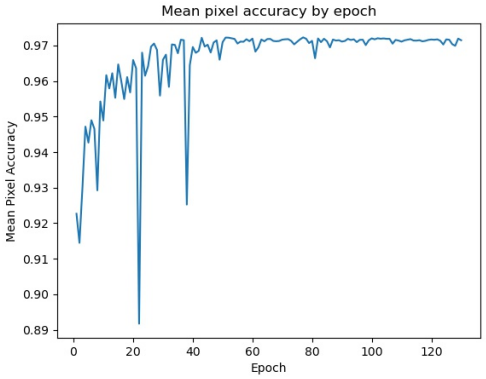
(e) Mean IoU evolution for 3^{rd} test.



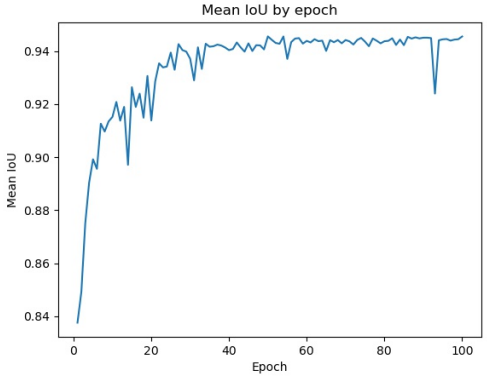
(f) Mean pixel accuracy evolution for 3^{rd} test.



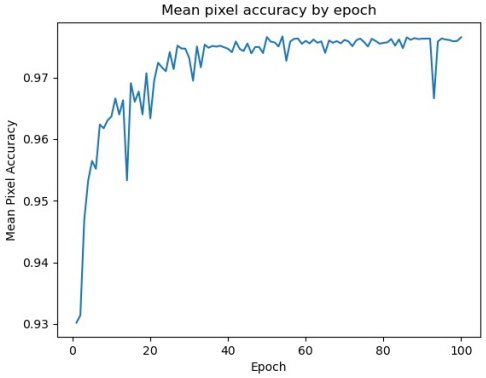
(g) Mean IoU evolution for 4^{th} test.



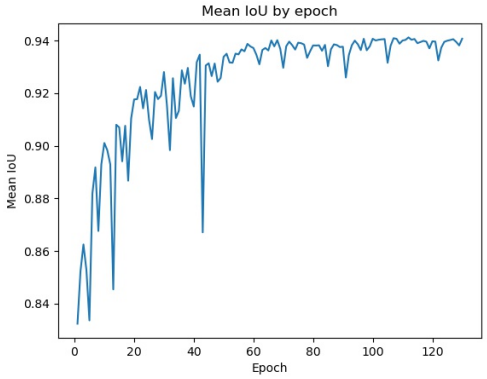
(h) Mean pixel accuracy evolution for 4^{th} test.



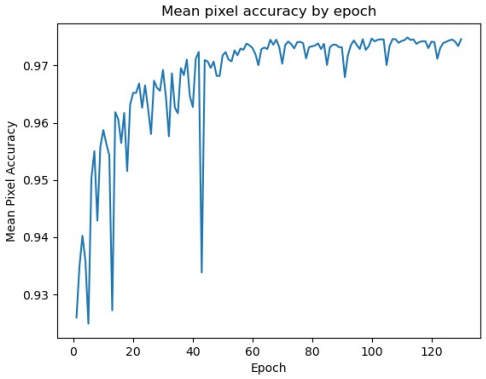
(i) Mean IoU evolution for 5^{th} test.



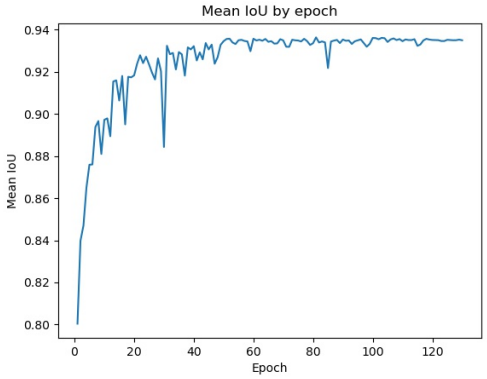
(j) Mean pixel accuracy evolution for 5^{th} test.



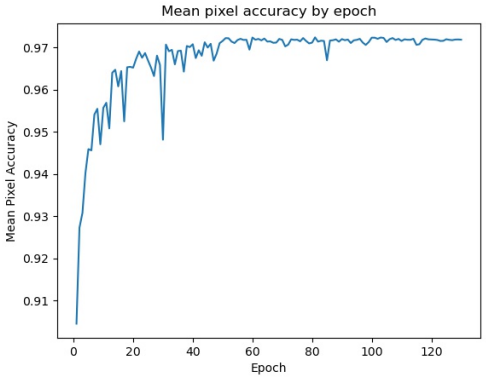
(k) Mean IoU evolution for 6^{th} test.



(l) Mean pixel accuracy evolution for 6^{th} test.



(m) Mean IoU evolution for 7^{th} test.



(n) Mean pixel accuracy evolution for 7^{th} test.

Figure 4.2: Evolution of the performance metrics, IoU and pixel accuracy, across epochs for each test.

4.1.2 Final Test

For the last test, the model used was the one reached when training for test number 7. The dataset used for this test was significantly different from the one used for training, since the images were collected in a different underwater cite, with different light conditions. The goal was to evaluate the model's capacity to generalize when conditions are different.

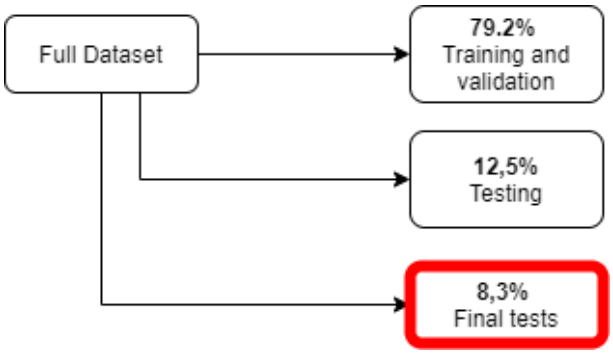


Figure 4.3: Dataset management chart. The red box denotes the part of the dataset used for the test explained in this section.

Pixel Accuracy	Mean IU
92.80%	85.21%

Table 4.4: Results achieved on the test set highlighted in figure 4.3

Looking at table 4.4, the network achieved very accurate results, with nearly 93% mean pixel accuracy and 85.21% IoU. The Sobel-based method achieved 91.5% mean pixel accuracy over the same samples. Although the FCN's accuracy decreased, comparing to previous tests, we can affirm that it is still a very accurate result. This high accuracy can be attributed to several factors

- The FCN is composed by part of a VGG16 network, which was previously trained on a very large general purpose dataset. Thus, it can contribute to good generalization and ease of learning.
- The FCN is usually used in problems with more than two classes. In this problem we only have two: soil foreground and background. Which becomes a simple task for an FCN to solve.
- Although the total dataset is composed of 1200 pairs of images and segmentation ground-truths, this is still a small dataset without much variety. Images are all very textured and, although some examples may be quite different, the vast majority of examples share numerous features.

Another interesting metric, the FCN can yield segmentation predictions in around 0.01 seconds, which translates to 100 Hz. In detail, this approach is able to generate 100 segmentation masks per second. The video footage used to create the dataset for this thesis had a frame rate of 25 frames per second. Meaning that our approach is 4 times faster than the frame rate of the camera. This gives us enough time to have segmentation predictions for every frame and allows a AUV for real-time image based navigation.

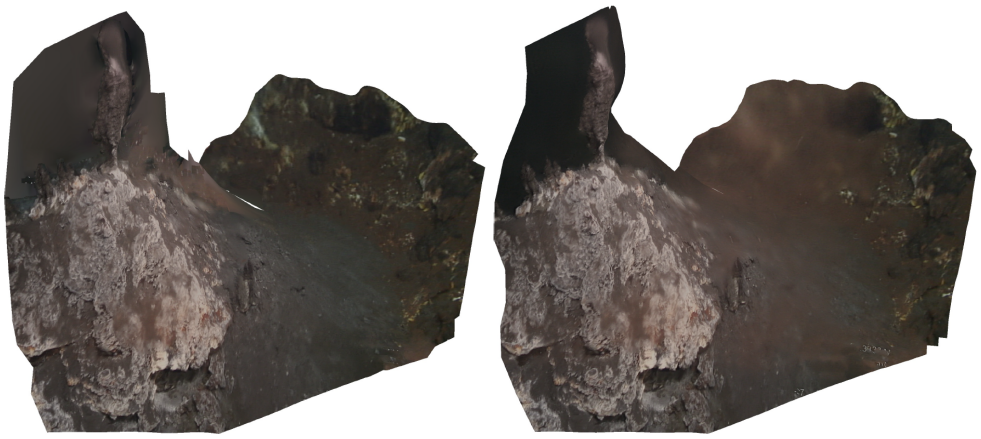
4.2 Simultaneous Localization and Mapping

To evaluate the impact of the proposed methodology on real time navigation algorithms we have ran offline SLAM and online SLAM tools on sequences of images with no masks and with masks generated by the trained FCN model.

4.2.1 Offline SLAM

Agisoft Metashape Standard (Version 1.7.2)(Software) is a photogrammetry software, used for 3-D reconstruction of scenes, being able to recreate textures. Up to a certain extent, much of what this tool does is offline SLAM, since it is used to process images after they were collected, with the intent of reconstructing camera poses and, therefore, the vehicle's trajectory.

To evaluate the impact of semantic segmentation on offline SLAM, the program ran a sequence of 46 images. Figure 4.4 displays the software's output with and without using masks. Although there is no huge difference between the two cases, we can see that when no masks are used 4.4 **(b)** the software builds the texture for a dust cloud that goes in front of the camera. The dust cloud has a brownish color. Furthermore, in the case without masks, the program also puts alpha-numerical characters, that appear on the corners of the input images, on the scene. When masks are used, neither the dust cloud, nor the characters are built on the reconstructed scene.



(a) (b)

Figure 4.4: **(a)** Output of Agisoft Metashape after processing a sequence of images with masks; **(b)**Output of Agisoft Metashape after processing a sequence of images without masks.

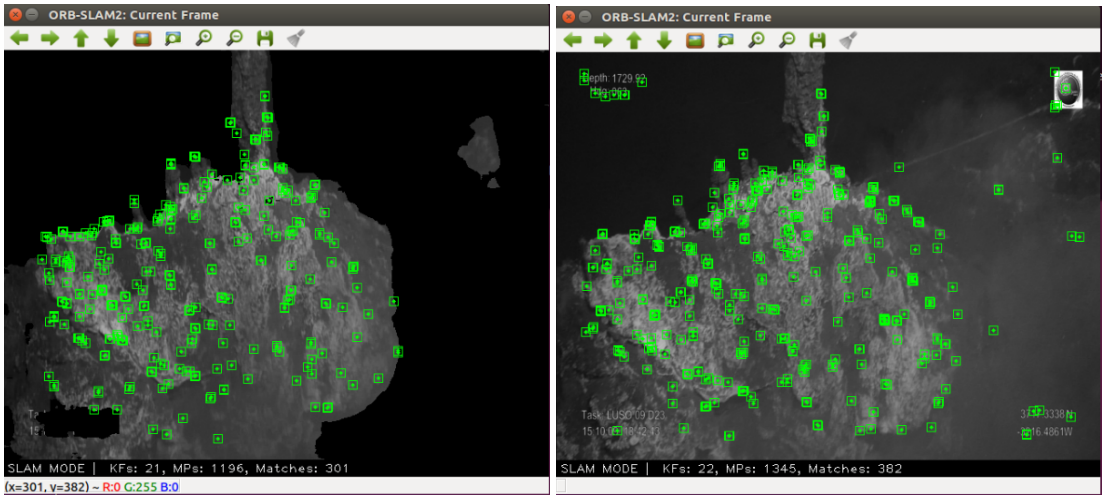
Table 4.5 shows some quantitative data, retrieved from a report generated by Agisoft. Comparing the two cases, there is no major differences between the two cases, which may be due to the software's robustness.

	With Mask	Without Mask
Run Time	8 minutes	7 minutes 39 seconds
Accumulated Memory Usage	2.42 GB	2.23 GB
Max Reprojection Error	1.13	1.13

Table 4.5: Comparison of some quantitative outputs of Agisoft Metashape.

4.2.2 Online SLAM: ORB-SLAM

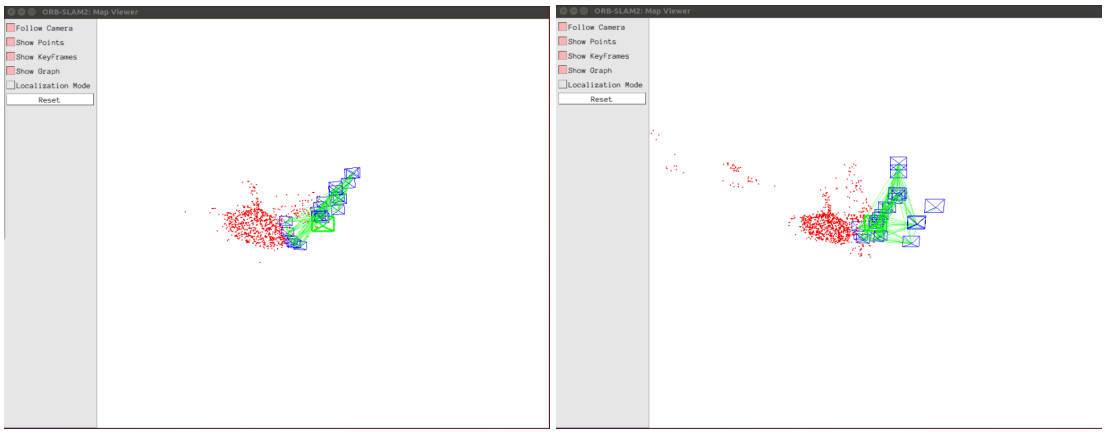
On the scope of online SLAM, ORB-SLAM was ran on a sequence of 1000 images. To evaluate the impact of using masks, the images and their respective masks were overlayed so that only the foreground was visible and everything else was 'deleted', by making every non-foreground pixel black, see figure 4.5 **(a)**.



(a) (b)

Figure 4.5: Frame with features extracted by ORB-SLAM **(a)** with mask; **(b)** without mask.

Figure 4.5 is screen capture of the current frame being processed by ORB-SLAM and the features it is tracking at the moment. Both were taken when a dust cloud partially covered the camera. On subfigure **(a)** we can see the mask erasing everything that is not a clear obstacle on the foreground and the algorithm only tracking features that belong to the object of interest. However, on subfigure **(b)**, where no masks were used, the algorithm tracks features that are not from rigid terrain, instead they belong to the dust cloud. Still on subfigure **(b)**, we can also see the algorithm tracking features that belong to regions of alpha-numerical characters. Over the course of several frames, ORB-SLAM will begin placing these points, that do not constitute sea bed or any form of terrain, on the map of the explored cite. We can see this in figure 4.6, comparing subfigures **(a)** and **(b)**, the last, which is the case where no masks are used, shows a lot of outliers. These outliers, points that are separated from what is the main foreground object, may come from features detected on the dust cloud and alpha-numerical characters.



(a) (b)

Figure 4.6: Maps generated by ORB-SLAM **(a)** with mask; **(b)** without mask.

Jerk

Jerk is a physics concept that amounts to the rate at which an object's acceleration changes over time. It is denoted by j and its units are m/s^3. It is usually expressed as a vector, being the first derivative of acceleration. It can also be expressed as a third derivative of position or a second derivative of velocity.

$$j(t) = \frac{d^3r(t)}{dt^3} = \frac{d^2v(t)}{dt^2} = \frac{da(t)}{dt} \tag{4.1}$$

Since ORB-SLAM outputs, for every keyframe, the coordinates of the camera's optical center, in respect to the world's coordinate system, jerk will be a 3-dimensional vector at any given point in time.

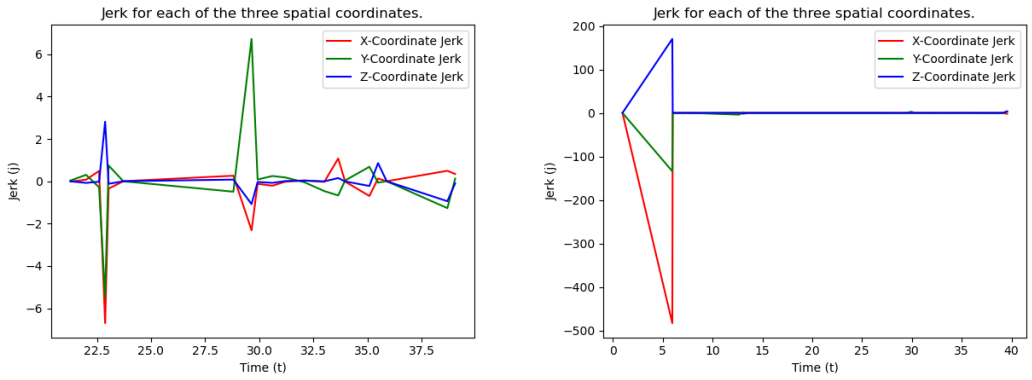
$$j(t) = \left(\frac{d^3x(t)}{dt^3}; \frac{d^3y(t)}{dt^3}; \frac{d^3z(t)}{dt^3} \right) \tag{4.2}$$

Besides the coordinates output, ORB-SLAM also provides the timestamps of those keyframes. This allows us to compute these derivatives through finite differences.

$$r'''(t_0) = \frac{-\frac{1}{2}r(t_{-2}) + r(t_{-1}) - r(t_{+1}) + \frac{1}{2}r(t_{+2})}{h_t^3} \tag{4.3}$$

where h_t is the time difference between each finite difference interval.

In robotics, a low jerk means the trajectory is smooth. In terms of the robot's control system, a smoother trajectory allows for less complex and more robust performance. Figure 4.7 and table 4.6 establish a comparison between the trajectory's jerk using masks and without masks. For a more accurate analysis, it should be noted that when running with segmentation masks, ORB-SLAM takes more time to initialize, since it detects less features per frame. However, it is able to find the same amount of keyframes (21) in a much smaller period. The results on table 4.6 were computed by calculating the average of all jerks' norms. When running with masks, the jerk is approximately 12 times better than without masks.



(a) (b)

Figure 4.7: Trajectory's jerk **(a)** with mask; **(b)** without mask.

With Mask	Without Mask
0.021	0.258

Table 4.6: Average of jerk norms.

4.3 Results Analysis

The results of this approach were exposed in two categories. To begin, we evaluate the performance of a neural network designed for semantic segmentation tasks and establish a brief comparison to classical methods. The next stage was to study the impact of this segmentation masks on a SLAM algorithm typically used in vehicles and robots capable of navigating autonomously. On an analysis of the achieved results, we could highlight the following findings:

1. The FCN yields good results for the problem of underwater image classification. With accuracy values above 90% and intersections over union around 85%, while taking only $0.01s$ to deliver a prediction;

2. Although the FCN was trained with a small dataset, the good performance proves that using a VGG net previously trained in a general purpose dataset, as an aproach of transfer learning, does have a positive impact on the overall results;

3. Comparing the results of the FCN and the Sobel operator, which was the best of the classical methods on a qualitative analysis, the former outperforms the latter;

4. The segmentation masks produced by the FCN improve the performance of offline SLAM algorithms, by avoiding the interference of dust and other floating particles during the 3D reconstruction of the environment surrounding the ROV;

5. On the online SLAM side, using the masks outputted by the FCN prevents the SLAM algorithm from identifying features that do not belong to foreground and thus should not be inserted to the

map generated by the algorithm. Examples of these unwanted features would be floating particles, blur and the characters present on the corners of the images. On a more quantitative metric, the trajectory's jerk is 12 times lower using masks compared to without masks, which suggests that ORB-SLAM algorithm identifies a smoother trajectory when using the results of the segmentation network.

Chapter 5

Conclusions

The purpose of this work was to study the impact of introducing AI-based semantic segmentation tools on the performance of SLAM algorithms, for posterior integration in autonomous underwater vehicles. An annotated dataset, with 1200 pairs of images and masks, was created to train a neural network to perform semantic segmentation and the results were tested on both online and offline SLAM algorithms.

To develop the dataset, images were enhanced to improve feature detection. Also, a tool was developed to yield an initial estimate of segmentation mask, using Sobel-based edge detection, which were hand corrected afterwards to create the segmentation ground truth for each image.

The chosen deep learning model for semantic segmentation was a Fully Convolutional Network, that was trained several times with different hyperparameters. With the network that yielded the best results, at around 93% accuracy and 85% IoU, a sequence of frames was ran through the model to have masks to test on the SLAM algorithms.

The results were tested on offline SLAM, using Agisoft Metashape Standard (Version 1.7.2)(Software) and on online SLAM, with ORB-SLAM. On both cases the proposed methodology improved the quality of the results. In offline SLAM, since the software is already very robust and well optimized, the core differences reside in the quality of the scene reconstruction. In the case of ORB-SLAM, using masks proved to improve the robustness of the model, by aiding in the feature tracking. Still on ORB-SLAM, the reconstructed trajectory using masks also yielded a lower jerk, which is an indicator of smoother and more robust navigation.

Future work should include the development of a larger and more diverse dataset and using it to train a Fully Convolutional Network, so the model can cope with conditions different from those of the proposed dataset. Also, for further validation, this methodology should be tested using an underwater autonomous vehicle on the context of real time ocean exploration.

Bibliography

[1] A. Sahoo, S. K. Dwivedy, and P. Robi. Advancements in the field of autonomous underwater vehicle. *Ocean Engineering*, 181:145–160, 2019. ISSN 0029-8018. doi: https://doi.org/10.1016/j.oceaneng.2019.04.011. URL https://www.sciencedirect.com/science/article/pii/S0029801819301623.

[2] L. Stutters, H. Liu, C. Tiltman, and D. J. Brown. Navigation technologies for autonomous underwater vehicles. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(4):581–589, 2008. doi: 10.1109/TSMCC.2008.919147.

[3] P. L. N. Carrasco, F. Bonin-Font, M. M. Campos, and G. O. Codina. Stereo-vision graph-slam for robust navigation of the auv sparus ii. *IFAC-PapersOnLine*, 48(2):200–205, 2015. ISSN 2405-8963. doi: https://doi.org/10.1016/j.ifacol.2015.06.033. URL https://www.sciencedirect.com/science/article/pii/S2405896315002724. 4th IFAC Workshop onNavigation, Guidance and Controlof Underwater VehiclesNGCUV 2015.

[4] S. Liu, M. Ozay, T. Okatani, H. Xu, K. Sun, and Y. Lin. Detection and pose estimation for short-range vision-based underwater docking. *IEEE Access*, 7:2720–2749, 2019. doi: 10.1109/ACCESS.2018.2885537.

[5] G. Allibert, M.-D. Hua, S. Krupínski, and T. Hamel. Pipeline following by visual servoing for autonomous underwater vehicles. *Control Engineering Practice*, 82:151–160, 2019. ISSN 0967-0661. doi: https://doi.org/10.1016/j.conengprac.2018.10.004. URL https://www.sciencedirect.com/science/article/pii/S0967066118306312.

[6] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[8] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *International Conference on Learning Representation (ICLR)*, 2015.

[9] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[10] F. Sultana, A. Sufian, and P. Dutta. Evolution of image segmentation using deep convolutional neural network: A survey. *Knowledge-Based Systems*, 201-202:106062, Aug 2020. ISSN 0950-7051. doi: 10.1016/j.knosys.2020.106062. URL http://dx.doi.org/10.1016/j.knosys.2020.106062.

[11] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[12] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[13] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham, 2015. Springer International Publishing.

[14] V. Badrinarayanan, A. Kendall, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(12):2481–2495, 2017. doi: 10.1109/TPAMI.2016.2644615.

[15] H. Noh, S. Hong, and B. Han. Learning deconvolution network for semantic segmentation. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[16] S. Umaa Mageswari, M. Sridevi, and C. Mala. An experimental study and analysis of different image segmentation techniques. *Procedia Engineering*, 64:36–45, 2013. ISSN 1877-7058. doi: https://doi.org/10.1016/j.proeng.2013.09.074. URL https://www.sciencedirect.com/science/article/pii/S1877705813015889. International Conference on Design and Manufacturing (IConDM2013).

[17] L. Chan, I. Morgan, H. Simon, F. Alshabanat, D. Ober, J. Gentry, D. Min, and R. Cao. Survey of ai in cybersecurity for information technology management. In *2019 IEEE Technology Engineering Management Conference (TEMSCON)*, pages 1–8, 2019. doi: 10.1109/TEMSCON.2019.8813605.

[18] B. Buchanan. *Artificial intelligence in finance*. The Alan Turing Institute, 2019. doi: 10.5281/zenodo.2626454.

[19] K.-H. Yu, A. L. Beam, and I. S. Kohane. Artificial intelligence in healthcare. In *Nature Biomedical Engineering*, volume 2, 2018. doi: 10.1038/s41551-018-0305-z.

[20] M. Roos. Deep learning neurons versus biological neurons. https://towardsdatascience.com/deep%2Dlearning%2Dversus%2Dbiological%2Dneurons%2Dfloating%2Dpoint%2Dnumbers%2Dspikes%2Dand%2Dneurotransmitters%2D6eebfa3390e9, 2019.

[21] N. Buduma and N. Locascio. *Fundamentals of Deep Learning*. O'Reilly Media Inc., USA, 2017.

[22] F. Bre, J. Gimenez, and V. Fachinotti. Prediction of wind pressure coefficients on building surfaces using artificial neural networks. *Energy and Buildings*, 158, 11 2017. doi: 10.1016/j.enbuild.2017.11.045.

[23] G. Hinton, D. Rumelhart, and R. Williams. Learning representations by back-propagating errors. In *Nature*, volume 323, pages 533–536, 1986.

[24] A. Swaminathan. Google ai summer school series lecture 5. https://archana1998.github.io/post/rahul-sukthankar/, 2020.

[25] P. Kim. *MATLAB Deep Learning*. Apress, Berkeley, CA, 2017.

[26] A. Deshpande. A beginner's guide to understanding convolutional neural networks part 2. https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/, 2016.

[27] Vgg16 – convolutional network for classification and detection. https://neurohive.io/en/popular-networks/vgg16/.

[28] H. Noh, S. Hong, and B. Han. Learning deconvolution network for semantic segmentation. In *Computer Vision (ICCV), 2015 IEEE International Conference on*, 2015.

[29] D. Mishra. Transposed convolution demystified. https://towardsdatascience.com/transposed%2convolution%2demystified%284ca81b4baba, 2020.

[30] Momentum and learning rate adaptation. https://www.willamette.edu/~gorr/classes/cs449/momrate.html.

[31] K. Weiss, T. M. Khoshgoftaar, and D. Wang. A survey of transfer learning. *Journal of Big Data*, 3, 2016. doi: 10.1186/s40537-016-0043-6.

[32] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010. doi: 10.1109/TKDE.2009.191.

[33] S. Thrun. *Simultaneous Localization and Mapping*, pages 13–41. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-75388-9. doi: 10.1007/978-3-540-75388-9_3. URL https://doi.org/10.1007/978-3-540-75388-9_3.

[34] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332, 2016. doi: 10.1109/TRO.2016.2624754.

[35] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós. Orb-slam: A versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015. doi: 10.1109/TRO.2015.2463671.

[36] K. He, J. Sun, and X. Tang. Single image haze removal using dark channel prior. *IEEE transactions on pattern analysis and machine intelligence*, 33(12), Dec. 2011.

[37] C. Ancuti, C. O. Ancuti, and C. D. Vleeschouwer. D-hazy: A dataset to evaluate quantitatively dehazing algorithms. *IEEE International Conference on Image Processing*, 2016.

[38] T. P. Marques, A. B. Albu, and M. Hoeberechts. Enhancement of low lighting underwater images using dark channel prior and fast guided filters. *Lecture Notes in Computer Science*, 11188, 2018.

[39] K. He, J. Sun, and X. Tang. Guided image filtering. *IEEE transactions on pattern analysis and machine intelligence*, 35(6), Dec. 2013.

[40] H. Kekre and S. Gharge. Image segmentation using extended edge operator for mammographic images. *International Journal on Computer Science and Engineering*, 2, 07 2010.

[41] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, 1986.

[42] N. Dhanachandra, K. Manglem, and Y. J. Chanu. Image segmentation using k -means clustering algorithm and subtractive clustering algorithm. *Procedia Computer Science*, 54:764–771, 2015. ISSN 1877-0509. doi: https://doi.org/10.1016/j.procs.2015.06.090. URL `https://www.sciencedirect.com/science/article/pii/S1877050915014143`.

[43] V. Puzyrev. Deep learning electromagnetic inversion with convolutional neural networks. *Geophysical Journal International*, 218(2):817–832, 05 2019. ISSN 0956-540X. doi: 10.1093/gji/ggz204. URL `https://doi.org/10.1093/gji/ggz204`.

[44] X. Liu, Z. Deng, and Y. Yang. Recent progress in semantic image segmentation. In *Artificial Intelligence Review*, volume 52, pages 1089–1106. Springer International Publishing, 2019.

[45] M. A. Rahman and Y. Wang. Optimizing intersection-over-union in deep neural networks for image segmentation. In G. Bebis, R. Boyle, B. Parvin, D. Koracin, F. Porikli, S. Skaff, A. Entezari, J. Min, D. Iwai, A. Sadagic, C. Scheidegger, and T. Isenberg, editors, *Advances in Visual Computing*, pages 234–244, Cham, 2016. Springer International Publishing. ISBN 978-3-319-50835-1.