

Performance evaluation of semantic segmentation in subsea conditions using AI trained on virtual images

Master's Thesis Project in Sustainable Energy Engineering with Specialization in Offshore Energy Systems

Miguel Ozalla Sánchez





Department of Energy Technology
Aalborg University
<http://www.aau.dk>

AALBORG UNIVERSITY

STUDENT REPORT

Title:

Performance evaluation of semantic segmentation in subsea conditions using AI trained on virtual images

Theme:

Master's thesis

Project Period:

Spring 2025

Project Group:

OES-10

Participant(s):

Miguel Ozalla Sánchez

Supervisor(s):

Jesper Liniger
Christian Mai

Page Numbers: 113**Date of Completion:**

June 2, 2025

The content of this report is freely available, but publication (with reference) may only be pursued due to agreement with the author.

Preface

This document constitutes the Master's Thesis for the Sustainable Energy Engineering program with a specialization in Offshore Energy Systems at Aalborg University.

The work was carried out during the spring semester of 2025, from February to June.

The project was supervised by Jesper Liniger and Christian Mai, under whose academic oversight the thesis was carried out. Acknowledgements are extended to both for their support and guidance during the project.

Abstract

Inspection and monitoring are critical activities in the maintenance and protection of underwater infrastructure. The opacity and turbidity of water can difficult the visibility of essential elements in such structures. This project explores the use of synthetic data generation to train a DeepLab v3+ semantic segmentation network with a ResNet-18 backbone. The objective of this thesis is to evaluate the neural network's ability to correctly identify objects in submerged environments with different turbidity levels. 1,800 image pairs comprising raw images and their corresponding segmentation masks were rendered by parameterizing water scattering and absorption in Blender. They were analyzed for five different situations, covering three discrete turbidity levels, an environment with no turbidity or diffraction elements, similar to the absence of water, and the case where all the aforementioned turbidity conditions and different water colorations are taken into account. The network was trained separately on each dataset and then evaluated on a turbidity test set to determine how sensitivity to training conditions affects real-world performance. Comparative analysis with models trained with low, medium, and high turbidity studies the impact of data diversity on prediction accuracy. These results offer valuable insights for developing robust vision systems for industrial applications such as pipeline inspection or structural health studies of infrastructure, where reliable object detection under turbid waters is essential for safety and operational efficiency.

Contents

1	Introduction	1
1.1	Underwater environments	1
1.2	State of the Art and Synthetic Image Use	2
1.3	Project Motivation	3
1.4	Project Objective	3
1.5	Research Pathline	4
2	Deep Learning	8
2.1	Resnet-18	9
2.2	Deeplab v3+	11
3	Experiment setup	12
3.1	Equipment	12
3.2	Overview of conducted experiments	19
3.3	Absorption and Scattering in the Experimental Pool	20
3.4	Visual Results from Laboratory Testing	24
4	Synthetic Images	27
4.1	Mesh modeling	28
4.2	Object's Material	30
4.3	Light Modeling in Blender	36
4.4	Water Volume Properties	37
4.5	Render Properties	39
4.6	Camera Pose Sampling	40
4.7	Synthetic Image Pair Creation	41
4.8	Other simulated environments	42
5	Convolutional Neural Network in matlab	43
5.1	Image preprocessing	43
5.2	Trainig of ResNet-18	49
5.3	Segmentation mask via DeepLab v3+	51
6	Results	56
6.1	No-Water Model	57
6.2	Low-Turbidity Model	61
6.3	Medium-Turbidity Model	65
6.4	High-Turbidity Model	69
6.5	Full Dataset Model	72

7	Discussion	77
7.1	Comparison between neural networks	77
7.2	Impact of turbidity on training	80
8	Conclusion	81
9	FutureWork	82
	Bibliography	83
A	Appendix - ResNet-18 Architecture	88
B	Appendix - Segmentation Mask Generation	91
C	Appendix - Training Procedure Using ResNet-18	96
D	Appendix - Semantic Segmentation Using DeepLab v3+	98
E	Appendix- Results	107
F	Appendix - Frame Extraction Script	113

1 Introduction

Over the last century, the society and industry in Europe have developed together. Many laws have been implemented to guarantee citizens' security and workers' rights. Precisely in Denmark, the beginning of the welfare state and worker protection flourished at the end of the 19th century, as outlined in [1]. This transformation occurred in two main stages.

Consolidation of the Welfare State (1930s)

In 1933, the Social Assistance Act ("Socialhjælpsloven") occurred, establishing public aid for families at risk and the unemployed. Later, in 1938, another law was published requiring compensation for workers in case they suffer work-related injuries. (Lov om Arbejdsskadeforsikring). These improvements established the beginning of a state focused on universal social protection [2].

Safety in Marine and Offshore Environments (1970s)

Following the discovery of oil fields in the North Sea around 1972, and the rapid development of offshore rigs, the Danish government began developing labor regulations in this sector. In 1977, a law was implemented requiring a reduction in the number of dangerous manual tasks and mandatory protective equipment [3], [4].

These two legislative stages were key in reducing exposure to dangerous tasks and increasing social and labor rights in Denmark. For this reason, over the past decades, the inspection and maintenance of offshore infrastructure has required the use of more specific technologies. This project focuses on the use of underwater vision technologies. The complexity of these environments is determined by several factors.

1.1 Underwater environments

Approximately a 70% of the Earth's surface is covered by water, it is understandable that underwater environments are incredibly diverse. Physical factors such as temperature, salinity, and dissolved oxygen concentration can generate completely different scenarios for both marine flora and fauna, and geological makeup plays a very important role in water conditions. This project studies optical properties such as water absorption and scattering, both of which vary with wavelength.

Other elements that affect water turbidity are sediments and dissolved or soluble organic matter in the aquatic environment itself. Generally, locations such as ports, where there is a sudden change from aquatic to terrestrial environments, are greatly affected by high turbidity levels. For the experimental period, a small pool will be used as a test bed

to monitor the turbidity levels studied. In general, it is important to account for water color, as it affects turbidity, and the colors in the chromatic range complicate segmentation based on different color thresholds. As discussed by Schechner and Karpel, the color of water significantly influences turbidity perception and complicates image segmentation based on color thresholds, especially in turbid environments [5].

Submerged elements can suffer from corrosion, like metals, or degradation and rot, like wood. Both materials have varied considerably refractive indices, so when combined with the aquatic environment, affect identification. Due to the difficulty and scarcity of large volumes of classified images in underwater environments, the idea of generating synthetic data in Blender arose. If the use of virtual images to identify objects in real environments proves successful, scalability to industrial applications will expand in the coming years.

1.2 State of the Art and Synthetic Image Use

Several recent studies have explored the use of deep learning for underwater object recognition and segmentation. Most of these approaches rely on convolutional neural networks (CNNs) such as U-Net or DeepLabv3+, which have shown excellent results in standard segmentation tasks [6]. However, a key limitation is the lack of annotated underwater datasets with sufficient variability in lighting, turbidity, and background clutter.

To overcome this limitation, researchers have turned to synthetic data generation. By simulating underwater scenes using tools like Blender or Unity, it is possible to create thousands of labeled images with controlled variations in geometry, lighting, and materials. The URPC [7] and the SUIM dataset [8] have contributed partially synthetic data for underwater vision research. Islam [9], for example, proposed a dual approach combining image enhancement with semantic segmentation to improve underwater perception. Similarly, Shafique [10] focused on training deep learning models exclusively on synthetic data to evaluate their generalization to real underwater scenes. These studies support the growing evidence that synthetic datasets can effectively supplement limited real-world data.

Mai et al. [11] proposed a methodology to generate synthetic underwater datasets with realistic lighting and marine growth using Blender, aiming to replicate real-world inspection scenarios. In a related study, Khan et al. [12] explored the use of these synthetic images for semantic segmentation tasks, demonstrating their effectiveness in training models capable of generalizing to real underwater conditions. These works represent a significant contribution toward bridging the domain gap between synthetic and real sub-sea imagery.

Despite the progress, few studies have quantitatively evaluated how well these models transfer to real subsea conditions, especially under varying degrees of turbidity, which

directly affects visibility and contrast. Most prior work focuses on clear-water environments or laboratory settings, where the domain gap is smaller.

1.3 Project Motivation

From an academic but industrially oriented perspective, a study on object identification under high levels of turbidity is considered interesting because it focuses on solving current problems that are being addressed at the industrial level. Its applicability in improving visibility in these environments is directly relevant to actions such as infrastructure maintenance, surveillance, and protection.

Different approaches can be used to address how turbidity and light attenuation complicate vision. Some examples include the use of powerful lights on remotely operated vehicles (ROVs) to ensure sufficient illumination, as well as the use of neural networks, where the decoder element strives to remove turbidity from these images and the identification algorithms are applied. Sensors such as sonars can also be used where turbidity does not significantly affect object identification. However, their resolution is very low, and they do not allow for the perception of small details or textures.

Convolutional neural networks (CNNs) were selected as an alternative approach, given the potential limitations of traditional inspection methods. The methodology adopted is described in detail in Chapter 1.5.

1.4 Project Objective

The main objective of the whole thesis is:

- To evaluate the performance of DeepLab v3+ with a ResNet-18 backbone for semantic segmentation in underwater environments with varying turbidity.

To achieve this main objective, various sub-objectives and tasks must be completed. These are:

- Generate sufficient synthetic images, with satisfactory quality and the ability to simulate different environments.
- Compare networks trained at different turbidity levels.
- Validate the networks and inspect their individual performance for the real laboratory images.
- Interpret real-world turbidity measurements and replicate them in a synthetic environment.

1.4.1 Delimitation

The main delimitations are listed:

- **Exclusive use of synthetic data:** All training and validation images are generated in Blender, without incorporating real-world field datasets beyond the final validation tests.
- **Limited number of classes:** Only five object categories are considered (pool background, black cylinder, silver cylinder, wooden piece, and granite), which does not cover the full diversity of underwater elements.
- **Controlled turbidity levels:** Four discrete turbidity grades are simulated (very low, low, moderate, and high), without modeling intermediate or dynamic variations.
- **Single network architecture:** Only DeepLab v3+ with a ResNet-18 backbone is employed, with no comparison to other segmentation topologies.
- **Static scenes:** All captures use a fixed camera and stationary objects; effects of motion, waves, or currents are not evaluated.
- **Homogeneous lighting and camera settings:** The same light properties (point/spot) and camera parameters are used throughout, without exploring alternative configurations or moving light sources.
- **No real-time processing:** The workflow focuses on offline rendering and segmentation; optimizations for inference on embedded systems or live video are not addressed.
- **Limited validation scope:** Validation with real images is restricted to laboratory tests; no trials are conducted in open-water or actual ocean conditions.
- **Limited time and resources:** Given the slightly over three-month timeframe, it was not possible to test alternative network architectures or different data-generation methodologies.
- **Camera connectivity and setup delays:** Significant issues were encountered when connecting the camera for real-image capture, leading to extended delays in preparing the system to a “ready-to-go” state.

1.5 Research Pathline

Most methodologies start with analyzing the research question and explain the relevance of such investigation. The current state of the case study needs to be researched. Subsequently a brainstorm is advised to define how to address and solve the problems presented. This entire process can be summarized as understanding and synthesizing the case study.

1. Introduction

The next step was to research different neural network structures (since these are networks designed to work with images). It is important to investigate how they work and what alternatives exist to see if it is worthwhile to use another strategy.

In the next stage, it is necessary to gather information on the generation of synthetic data, including the key aspects involved and the available tools for this purpose. Additionally, it is important to investigate methods for creating segmentation masks. A thorough review of all relevant topics should be conducted in order to enable subsequent analysis and evaluation of the results.

The last step is to develop a work plan of the points to be addressed, how to approach them, and estimate how much time will be needed for each phase of the project. Finally, the creation of a diagram that encompasses all the steps to be followed throughout the project.

The project is presented in Figure 1.1 and can be summarized as a block diagram. The main block, or process, is the CNN, which has two inputs and one output. All relevant steps during the project are introduced later in this section.

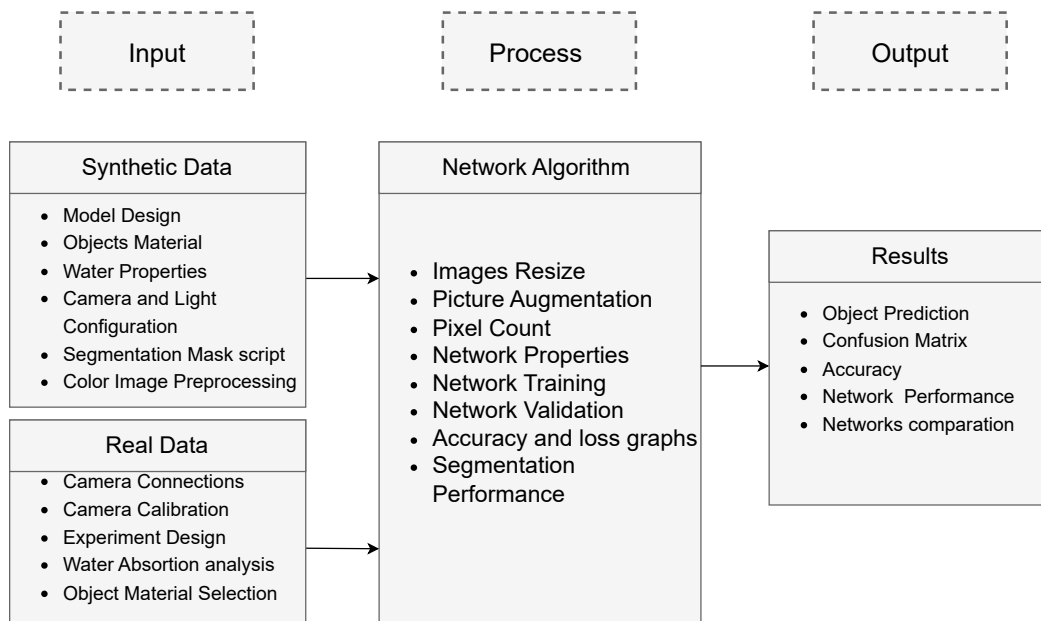


Figure 1.1: Research Pathline

Synthetic Data

One of the inputs corresponds to the synthetic data. To create a successful data package, several tasks must be performed. These are presented as bullet points and entered in the following list:

- **Model Design:** The mesh of all objects to be included in the render is designed.
- **Object's Material:** The material architectures to be assigned to each object are chosen.
- **Water Properties:** The absorbance and scattering properties of the water are established in Blender.
- **Camera and Light configuration:** The lighting and camera options are modified to match those of the experimental chamber and the laboratory lighting.
- **Segmentation Mask script:** A script is generated to obtain a segmentation mask for each generated render.
- **Image Preprocessing:** The generated mask images are resized, and color deviations are corrected.

The synthetic data is intended to feed the network for training and then validate how the network performs in correctly predicting classes.

Real Data

This section describes the procedure followed to obtain the real images used for testing and comparing the performance of the neural networks. These images are subsequently linked to the network evaluation process. A detailed explanation of each step involved in data acquisition and preparation is provided.

- **Camera Connections:** To execute a plan to feed the camera and extract real-time data transfer.
- **Camera Calibration:** To calibrate the camera's optical properties to obtain the best possible image quality.
- **Experiment Design:** Experiments were designed to include the desired variability in turbidity for the project.
- **Water Absorption Analysis:** Water turbidity was analyzed in the laboratory in order to translate it into synthetic data.
- **Object Material Selection:** Objects with distinct shapes and textures were selected to ensure varied light reflection and facilitate their identification.

Network Algorithm

DeepLab v3+ was chosen to perform the segmentation process, using the ResNet-18 network as the backbone, responsible for identifying groups of objects in images. To build the network, the code was divided into small sections with well-defined functions. Finally, the network produced a segmentation mask prediction and a copy of the input test image as output. The small operations that were executed are the following:

- **Image Resizing:** The input images were resized to ensure they had the correct dimensions.
- **Image Augmentation:** The training images were distorted within defined margins and randomly deformed to enhance the feature extraction process.
- **Pixel Count:** The pixels were counted to subsequently balance the class weights.
- **Network Properties:** The network training options were defined.
- **Network Training:** The network architecture was established and trained under the selected options.
- **Network Validation:** The network was validated using synthetic data to verify its performance.
- **Accuracy and Loss Graphs:** The accuracy and loss curves were generated.
- **Segmentation Performance:** The prediction was carried out on the test image.

Results

This is the final step, where the results are obtained. The network, under certain commands, provides metrics to evaluate its performance in addition to the predictions. Ideally, these results serve as the starting point for an iterative method where, based on the conclusions, small adjustments are made to both the network and the synthetic data to improve predictions and data generation. Due to time constraints, the knowledge and results obtained could not be used as feedback to improve the prediction process. The results obtained are listed below.

- **Object Prediction:** The predictions were obtained in the form of segmentation masks.
- **Confusion Matrix:** A matrix was generated to indicate the accuracy with which the classes were correctly identified.
- **Accuracy:** The accuracy and loss curves were plotted.
- **Network Performance:** The predictions were compared with the manually created segmentation masks from the experimental images.
- **Networks Comparison:** The results provided by each network were compared.

2 Deep Learning

Since the Industrial Revolution emerged in the late 18th century, the idea of automating tasks has grown. The advantages are clear: avoiding repetitive tasks, reducing production costs, and speeding up production processes. Centuries later, after the Second World War, the potential of computer science as a scientific discipline began to develop as stated in [13]. Since the 1940s, this discipline has advanced in the process of automating intellectual tasks. This process has advanced so much that it has now generated the concept of artificial intelligence, which is nothing more than a tool of algorithms and mathematical processes, whose objective is to design systems capable of reasoning, learning, and executing decisions autonomously.

Within artificial intelligence, there is the field of machine learning (ML), in which machines learn by feeding them data. This learning differs from conventional learning in that it does not provide manually written lines of code that dictate rules or orders. Within ML, there are several main techniques. On the one hand, there is supervised learning, where models receive labeled data; on the other, unsupervised learning, where unclassified data is provided and the model searches for patterns and extracts features. There is also reinforcement learning, where the model makes predictions through a system of rewards and punishments [14].

Deep learning (DL) emerges from ML when more complex models begin to be developed with greater depth and number of layers, allowing learning from more complex environments [15]. The architecture of deep learning models is typically structured across three distinct types of layers:

- **Low layers:** responsible for extracting basic and localized features, such as edges, textures, or simple patterns.
- **Middle layers:** integrate and combine lower-level features to identify more complex structures or shapes within the input data.
- **High layers:** abstract and consolidate information from the middle layers to recognize high-level representations, such as object categories, actions, or complete scenes.

There are many memorable events throughout history regarding the progress made, starting from the perceptron in 1958 [16] where the first artificial neuron capable of linear classification was created. In 2012 AlexNet emerged [17] where systems were developed to use GPUs during the training process and therefore increasing their speed for predicting. And in 2015, the concept of Residual Network emerged from Kaiming He's study called Deep Residual Learning for Image Recognition [18]. This study offers a quite useful solution to the problem of gradient reduction in the backforward process, which

provided an increase in error for models with more layers and which intuitively should provide significantly lower errors than similar models but with fewer layers.

As a hierarchical synthesis, Figure 2.1 shows the order from a more general level to reach the topic of computer vision, whose field of study focuses on, through digital images, being able to process tasks such as classification, detection, segmentation or 3D reconstruction.

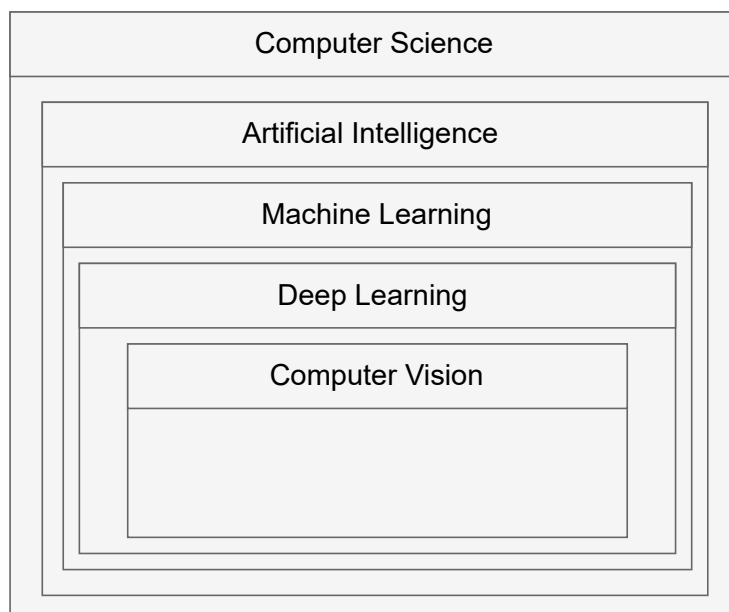


Figure 2.1: Hierarchy of Disciplines in Artificial Intelligence

2.1 Resnet-18

As time goes by, new, deeper and more powerful neural networks are constantly (CNN) emerging. CNN are a special type of neural network used to analyze images. The model used in this project is ResNet-18, proposed by [18]. This network consists of 18 layers organized into residual blocks. Previous versions of CNNs, such as LeNet-5 [19], AlexNet [17], VGGNets [20], or GoogLeNet/Inception v1 [21], faced problems with vanishing or exploding gradients during backpropagation. These gradients could become extremely small, causing the first layers to learn very slowly or even fail to learn at all. Conversely, when gradients became too large, they made the network unstable, providing very large weights where solutions could not converge [22].

Kaiming's network model offered solutions to these problems, allowing deeper networks to train successfully. The most significant contribution was the use of residual blocks with direct connections or shortcuts. In simple terms, rather than letting a stack of layers directly learn a transformation ($H(X)$), a bypass or shortcut is applied on that layer or set

of layers, and, an identity is added to the output of those layers, generally called x , forcing the layers to learn a residual function that would be $F(x) = H(x) - x$. This methodology solves the degradation problem, since these shortcuts provide shorter alternative paths for gradients to flow to previous layers. Allowing the addition of more layers, which generates deeper and more powerful networks.

2.1.1 Architecture

The MATLAB has been employed for the implementation of this network [23]. The 18 layers that make up the network are organized in blocks, where each layer would in turn have convolutional layers and residual shortcuts. The convolutional layers are those that extract the image features. These layers are made up of the size of the input image to the layer, the kernels used (3x3, 7x7, etc.), the applied stride, which indicates the kernel displacement, and the padding (additional pixels around the image). All of these elements define the output size of the layer. There are also pooling layers that modify the image resolution. The most notable feature of this network model is the residual connections it uses.

Regarding the general composition of the network, it begins with the Input layer, which is fed into a first convolutional layer to detect notable features, such as edges, colors, angles, etc.

There are 4 residual blocks, and each block is divided into 2 different layers, the purpose of which is to extract more complex patterns than those of the input layer. In these convolution blocks, there are convolution layers. As the signal passes through a layer, an activation function is applied to extract nonlinear features. By default, the activation function is ReLu (rectified linear unit). This function is responsible for setting the input signal value to zero if and only if the initial value was negative; otherwise, it does not modify the signal value. The organization at the filter or kernel level for the residual blocks is as follows:

1. In the first block, there are 64 filters.
2. In the second block, there are 128 filters.
3. In the third block, there are 256 filters.
4. In the fourth block, there are 512 filters.

After these blocks, the image passes to a pooling layer to reduce its size and is sent to a fully connected layer, which takes the final image and uses it to make a final decision. Finally, there is the output layer, which has a number of channels equivalent to the number of classes to be identified in the network. Appendix A shows an image A.1 with the block-level architecture of the network and a table specifying the contents of each block.

2.2 Deeplab v3+

ResNet-18, previously discussed as a convolutional neural network for image classification tasks, includes final layers that produce a probability vector across a set of predefined classes. The network outputs the class with the highest probability, which is interpreted as the predicted label. A correct classification is assumed when this predicted class corresponds to the actual object in the image.

In this section, the use of DeepLab v3+, a semantic segmentation architecture developed by Google in 2018, is presented. In this configuration, ResNet-18 functions as the encoder, extracting hierarchical features from the input image. These features are then processed by a decoder module—specifically the Atrous Spatial Pyramid Pooling (ASPP)—to generate a segmentation mask that classifies each pixel individually.

The network was trained using pairs of images: one showing the original scene and another providing a pixel-wise annotated mask, where each pixel corresponds to a specific object or class present in the composition.

As a main feature of this network, it highlights the use of ASPP, which employs dilated (atrous) convolutions, increasing the image's field of view without losing resolution. The network output is an image segmentation where the results have been refined to make them sharper and more accurate. Regarding optimization, the cross-entropy loss function is generally used, which calculates the difference between the predicted and actual labels for each pixel. This function is used by the Adam algorithm to adjust the network's weights during the training phase.

For the implementation of this network, Matlab was used due to familiarity with the tool. There is extensive documentation available online and examples similar to the case study. The main advantage of Deeplab v3+ is its accuracy; however, the model is computationally intensive. As for its applications, it has been used in fields such as the automotive industry to identify road elements, and in healthcare to detect injuries through X-rays or MRIs.

3 Experiment setup

The goal is to generate a synthetic dataset for simulating underwater environments, with the ultimate purpose of improving the identification of objects in environments with low visibility, whether due to turbidity, lack of light, or suspended particles that hinder visibility.

However, to measure the quality of this data, it is necessary to compare it with real scenes. In this case, different objects will be distributed in varying positions and orientations at different turbidity levels. A model will be prepared in the laboratory for this purpose. The experiments are carried out in a controlled environment. The measurements of the pool and all the objects used, are known. The Lighting information of the fluorescent lamps is extracted by the technical specifications of the product [24].

3.1 Equipment

This chapter explains all the elements used to carry out the experiment at the university facilities. This includes the objects to be identified and the camera connections, including how the turbidity of the pool water was manipulated and how it relates to the values entered into the virtual simulation.

3.1.1 Lab's Lightning

During the experimental phase, the lighting in the laboratory was constant. Fluorescent bulbs were used. The light used does not have a very marked direction, but rather envelops the scene like natural light, without generating shadows or very marked contrasts. The technical specifications of which are shown in 3.1. An image of the lighting is shown in figure 3.1.

3. Experiment setup



Figure 3.1: T5 fluorescent tube

Property	Value	Unit
Brand	Sylvania	–
Tube Type	T5	–
Power	80	W
Length	1450	mm
Diameter	16	mm
Luminous Flux	6150	lm
Color Temperature	3000	K
Light Tone	Warm White	–
Base Type	G5	–

Table 3.1: Main characteristics of the T5 fluorescent tube used in the experimentation [24]

3.1.2 Pool

The technical data sheet for the pool model [25] details its dimensions and materials. This data sheet was obtained through manual measurement and an internet search for the model printed on the pool's exterior surface. The pool has a filter system to recirculate and clean the water, which has never been used. For this reason, in later stages of the project, the water quality exhibits inherent turbidity without the addition of external elements, nevertheless, the turbidity will be increased externally. This turbidity occurs due to a lack of chlorine and ideal temperatures, as well as light, which can contribute to the growth of microalgae.



Figure 3.2: Bestway Power Steel Rectangular Pool

Property	Value	Unit
Brand	Bestway	–
Model	Power Steel	–
Shape	Rectangular	–
Dimensions	404 × 201 × 100	cm
Volume (90%)	6,478	L
Frame Material	Steel	–
Liner Material	Tritech™ (3-layer PVC)	–
Inner Finish	Mosaic Print	–
Color	Rattan Grey	–

Table 3.2: Main characteristics of the Bestway Power Steel pool used in the experimentation [25]

3.1.3 Object to identify

The selected components were chosen due to their diversity in size, color, and the way light interacts with their surfaces. Another key factor in their selection is their resemblance to commonly used industrial shapes, as well as to forms frequently found in nature. As a result, the initial elements consist of cylindrical and parallelepiped geometries. Specifically, the setup includes two pipes of different diameters, granite cobblestones, and a wooden table. Table 3.3 lists their dimensions, while Figure 3.3 illustrates their main characteristics.



Figure 3.3: Selection of objects to identify

Object	Dimension	Value [m]
Black Pipe	Length	0.30
Black Pipe	Radius	0.025
Aluminum Pipe	Length	0.95
Aluminum Pipe	Radius	0.050
Granite Cobblestone	Side Length	0.40
Wooden Piece	Width \times Length \times Height	$0.19 \times 0.30 \times 0.05$

Table 3.3: Main dimensions of the objects to be identified in the segmentation task

3.1.4 Kaolin

To increase the pool's turbidity, kaolin was used. The composition is found in 3.4. This clay mineral is used because it is not water-soluble, forming a suspension and not health hazardous. It is sprinkled into the water solution and mixed with the water using brushes, increasing its turbidity. The dissolved kaolin particles modify how light scatters in the water and give it a whitish color.

For each degree of turbidity, the amount of kaolin was increased, and samples were col-

3. Experiment setup

lected to subsequently analyze the total absorbance of the water in each experiment.



Component	Estimated Content
Kaolin	> 90%
Silica	45–55%
Alumina	35–40%
Iron oxide	< 1%
Titanium dioxide	< 2%
pH (in water)	4–6
Appearance	White powder

Figure 3.4: Kaolin and its representative composition [26]

3.1.5 Testing Environment

The experimental tests were conducted in a controlled environment at the AAU-Esbjerg laboratory. As shown in Figure 3.5. The pool setup allowed for the submersion of objects at different depths and positions under stable lighting conditions.

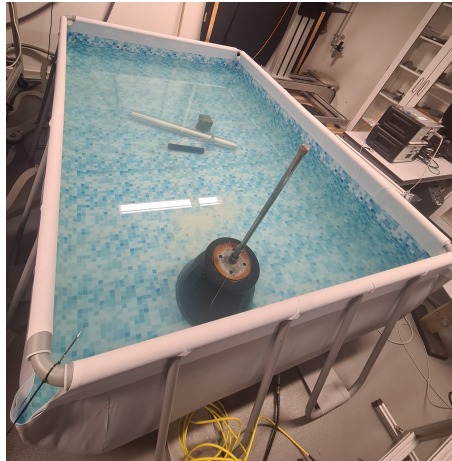


Figure 3.5: Lab-scale pool used for the experimental trials

3.1.6 Camera Connection and Configuration

The camera model used is illustrated in figure 3.6, it is the camera with the best quality and resolution available at the time of this project at the AAU-Esbjerg. The technical data sheet shows that its optical sensor is 1/2.9" SONY CMOS 2 Megapixel and that its optimal wavelength captured range is 500-700 [nm]. Therefore, the average value is taken as

3. Experiment setup

the starting wavelength to obtain the absorbance values of the analyzed samples.



Figure 3.6: Oceantools C3-30 Subsea Camera [27]

The camera power connector is an 8-pin interface, with each pin assigned to a specific function. As a first step, a multimeter was used to check continuity between the connector pins and the ends of the cable that would later interface with the Fathom Tether [28]. This test aimed to verify the integrity of the cable and confirm whether the camera was operational or exhibited any faults.

Using the camera's technical datasheet, the function of each pin was identified. Figure 3.11 illustrates the wiring between the camera and the tether. For proper pin-to-pair mapping between the 8-pin connector and the four twisted pairs of the Blue Robotics Fathom Tether (an Ethernet-type cable), the Advantech patch cable white paper was used as a reference [29], [30], as it details the standard RJ45 pair-to-terminal assignments.

















Pins at switch	T568A color	T568B color	10/100 mode B, DC on spares		10/100 mode A, mixed DC & data		1000 (1 gigabit) mode B, DC & bi-data		1000 (1 gigabit) mode A, DC & bi-data	
Pin 1	 White/green stripe	 White/orange stripe	Rx +		Rx +	DC +	TxRx A +		TxRx A +	DC +
Pin 2	 Green solid	 Orange solid	Rx –		Rx –	DC +	TxRx A –		TxRx A –	DC +
Pin 3	 White/orange stripe	 White/green stripe	Tx +		Tx +	DC –	TxRx B +		TxRx B +	DC –
Pin 4	 Blue solid	 Blue solid		DC +	Unused		TxRx C +	DC +	TxRx C +	
Pin 5	 White/blue stripe	 White/blue stripe		DC +	Unused		TxRx C –	DC +	TxRx C –	
Pin 6	 Orange solid	 Green solid	Tx –		Tx –	DC –	TxRx B –		TxRx B –	DC –
Pin 7	 White/brown stripe	 White/brown stripe		DC –	Unused		TxRx D +	DC –	TxRx D +	
Pin 8	 Brown solid	 Brown solid		DC –	Unused		TxRx D –	DC –	TxRx D –	

Figure 3.7: Power over Ethernet connections

3. Experiment setup

The required power to turn on the camera is 24 [V], the camera had to be powered with an external source. The injector [31] passively provides sufficient voltage from the beginning of the connection to launch the camera. Besides, the switcher [32] transfers and monitors the transmitted packages. Both devices are presented in figure with their respective references.



Figure 3.8: Switch and Injector

A port scan was performed on the computer to know the IP address connection. The employed software were SoftPerfect Network Scanner [33] and SADP [34]. Additionally the manufacturer (OceanTools [35]) was contacted to verify that the mounting system 3.9 was not harmful to the camera. Moreover, the manufacturer was contacted to consult the username and password required to get access to the camera settings.

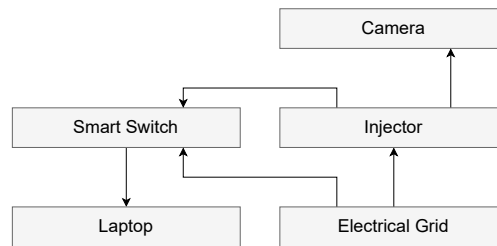


Figure 3.9: Camera Connection Architecture

Subsequently, each pair of cables was soldered and properly insulated. Once the physical connection was completed, and before attempting communication between the camera and the computer, a transmission test was carried out to evaluate the quality and integrity of data packets exchanged between both devices. After contacting the manufacturer, it was possible to successfully establish a connection with the camera and extract images.

3. Experiment setup

Figure 3.10 shows the evolution of the insulating process.



Figure 3.10: Cable Soldering & Waterproofing

To control the entire recording process, VLC Media player [36] to record and store the images, and then, using a Matlab script, we extracted the desired frames with the best image quality.

To manage the entire recording process, VLC Media Player [36] was used to capture and store the video streams. Subsequently, a MATLAB [37] script F was employed to extract the frames with the highest image quality from the recorded footage.

Camera calibration was performed through the camera's settings interface 3.11. To reduce noise caused by movement, the camera was secured to a metal post using safety ties, which fixed the camera's position and orientation relative to the water movement.

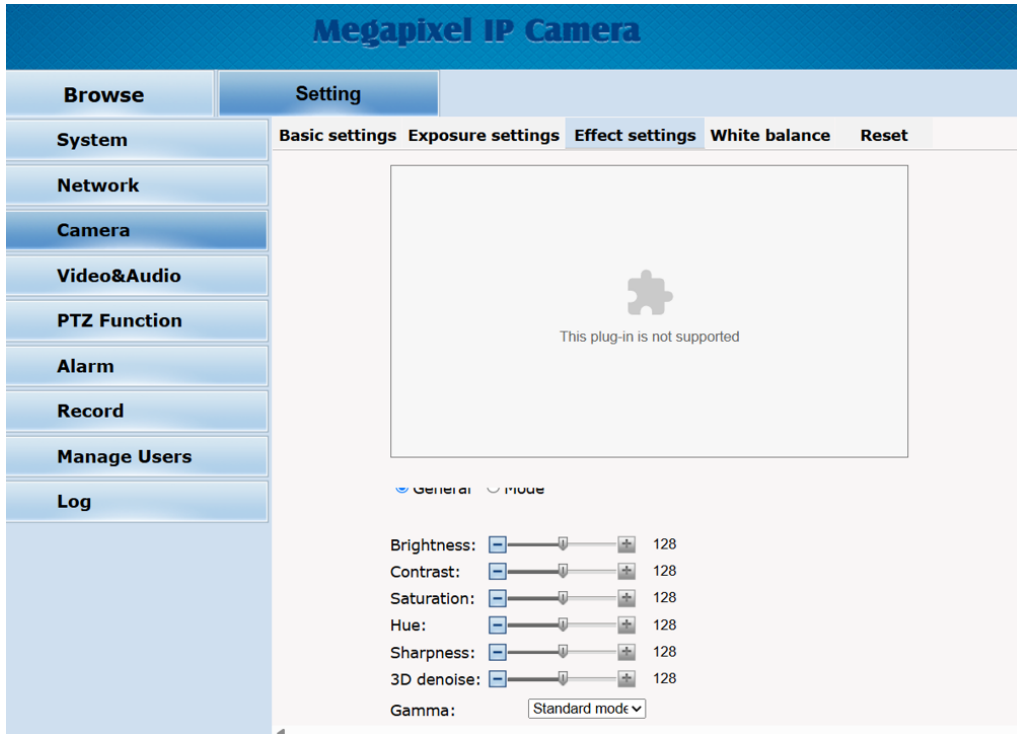


Figure 3.11: C3-30 Subsea Camera's interface

Various tests were performed throughout the calibration process to choose the camera configuration option with the values that provided the best image quality. Unfortunately, no samples were collected to be shown.

3.2 Overview of conducted experiments

The objective of the experiments is to evaluate the network's ability to accurately identify various objects in images affected by turbidity. To assess performance under different perspectives, two distinct object arrangements were created. For each arrangement, four experiments were conducted, each corresponding to a different level of turbidity.

The initial experimental plan included capturing an additional image set under clear water conditions, intended to serve as a baseline with zero turbidity. However, since the pool water was not chemically treated, it gradually developed a greenish coloration and a mild level of turbidity over time. This natural coloration was later masked by the addition of kaolin during the turbidity experiments. For this reason, the reference test does not display the same positioning as the other sets and includes an additional element, removed from the classes to be identified.

Regarding turbidity levels, four distinct degrees were defined, each applied to two different object arrangements. This resulted in a total of eight experiments.

1. Very low turbidity: kaolin is added and visibility remains good.

3. Experiment setup

2. Low turbidity: kaolin concentration increases and object silhouettes begin to appear blurred.
3. Moderate turbidity: visibility is reduced; although objects can still be recognized at a glance, many details are lost.
4. High turbidity: visibility is very poor; it becomes almost impossible to discern objects and their edges are heavily blurred.

The following image 3.12 shows a diagram used in the experimentation, additionally, the sequential process is explained. During this entire process the camera has been fixed without changing its orientation or position.

First the position and orientation of the objects to be shown were located and captured. A photo of their distribution was taken to replicate it later. For a given distribution, the photos are taken and the second arrangement of objects is made, these are recorded and kaolin is added to them while it is mixed manually and the entire process is recorded, to select a significant image of this experiment. When the desired degree of turbidity has been reached and the image stored, it is changed to the previous position and the scene is recorded again. This iterative process of taking experiments and changing the position of objects continues until all the desired combinations are achieved.

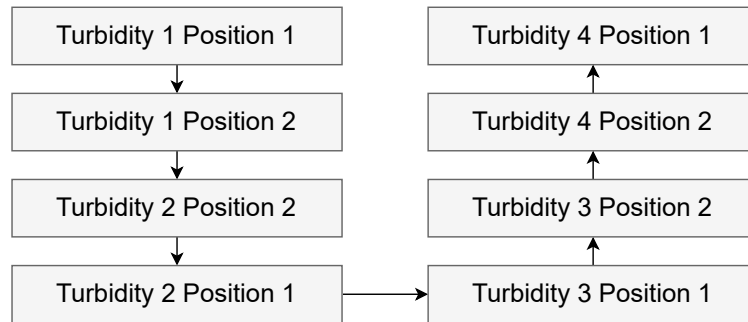


Figure 3.12: Experiment work flow

3.3 Absorption and Scattering in the Experimental Pool

In the pool with kaolin in suspension, the light that traverses the water is affected by the physical phenomena of absorption and scattering.

1. Absorption (μ_a), that describes the part of the luminous energy converted into heat.
2. Scattering (μ_s) is the change in direction upon colliding with particles, which makes the water blurry.

To quantify how the intensity $I(l)$ decays after traveling a distance l [m], It was started from the Beer–Lambert law [38], [39], equation 3.1 and 3.2 summarize the relations between intensity and apparent absorbance. Here the intensity is radiometric and measures

3. Experiment setup

the radiant power incident on a unit area after traveling a distance l . Its unit is watts per square meter $[\text{W}/\text{m}^2]$:

$$I(l) = I_0 \exp[-c_{\text{app}} l] \quad (3.1)$$

$$A_{\text{app}} = -\log_{10}\left(\frac{I(l)}{I_0}\right) \quad (3.2)$$

- $c_{\text{app}} = \mu_t$ is the total apparent attenuation coefficient.
- A_{app} is the apparent absorbance, the ratio between transmitted and incident irradiance; it explains in a dimensionless way how the medium darkens due to particle suspension. Higher values indicate more attenuation and therefore greater darkness/turbidity. The apparent absorbance is measured with the Cary 60 UV-Vis spectrophotometer [40].
- Incident irradiance I_0 is the radiant power per unit area before the light begins to traverse the distance l .
- Transmitted irradiance $I(l)$ is the radiant power that emerges after traversing the medium.

The factor 2.303 arises when converting between the natural logarithm (\ln) and the decimal logarithm (\log_{10}):

$$\ln(x) = 2.303 \log_{10}(x). \quad (3.3)$$

Solving for c_{app} gives

$$c_{\text{app}} = \frac{2.303 A_{\text{app}}}{l}. \quad (3.4)$$

The apparent attenuation coefficient c_{app} can be decomposed into its two fundamental physical components: absorption (μ_a) and scattering (μ_s). The ratio between these components is known as the scattering-to-absorption ratio, defined as $R = \mu_s/\mu_a$. According to previous studies conducted under controlled coastal and laboratory water conditions, this ratio typically ranges between 20 and 29 [38], [41]. These references, along with equations 3.5 and 3.6, describe the optical decomposition of light attenuation and the relationship between absorption and scattering coefficients.

$$c_{\text{app}} = \mu_a + \mu_s \quad (3.5)$$

$$\mu_s = R \mu_a, \quad (3.6)$$

Substituting equations 3.5 and 3.6, the following expressions are obtained

$$\mu_a = \frac{c_{\text{app}}}{1 + R}, \quad (3.7)$$

$$\mu_s = \frac{R c_{\text{app}}}{1 + R}. \quad (3.8)$$

Finally, by substituting the term c_{app} from equation 3.4 into equations 3.7 and 3.8, the resulting expressions are given in equations 3.9 and 3.10.

$$\mu_a = \frac{2.303 A_{\text{app}}}{l (1 + R)}, \quad (3.9)$$

$$\mu_s = \frac{2.303 R A_{\text{app}}}{l (1 + R)}. \quad (3.10)$$

D Building upon the theoretical foundations established in the previous section, the following subsection describes the procedure for obtaining water samples and measuring the apparent total attenuation coefficient. These measurements are subsequently used to derive the absorption (μ_a) and scattering (μ_s) coefficients.

3.3.1 Absorbance analysis

During the experimental phase, 15 [mL] conical centrifuge tubes, commonly known as Falcon tubes [42], were employed to collect water samples. These containers, depicted in Figure 3.13, enabled the storage of sufficient water volumes for multiple analyses per sample, thereby minimizing potential deviations. Each sample underwent four tests, with the average value representing the analysis.

The analyses were conducted using the Agilent Cary 60 UV-Vis Spectrophotometer [40], scanning each sample across a wavelength range from 300 [nm] to 800 [nm], recording absorbance at 5 [nm] intervals. This range adequately encompasses the human visible spectrum, which spans approximately from 380 [nm] to 700 [nm] [43].

For these measurements, plastic spectrophotometer cuvettes [44] were utilized. However, for analyses involving wavelengths below 300 [nm], it is recommended to use quartz cuvettes due to their superior transparency in the ultraviolet range [45]. The cuvette used has a path length of 1 [cm]. The apparent absorbance values obtained across the spectral range are presented in Figure 3.14, while the corresponding absorption and scattering coefficients at 640 [nm] are summarized in Table 3.4.

3. Experiment setup



Figure 3.13: Left plastic cuvette [44] and right experimental samples [42]

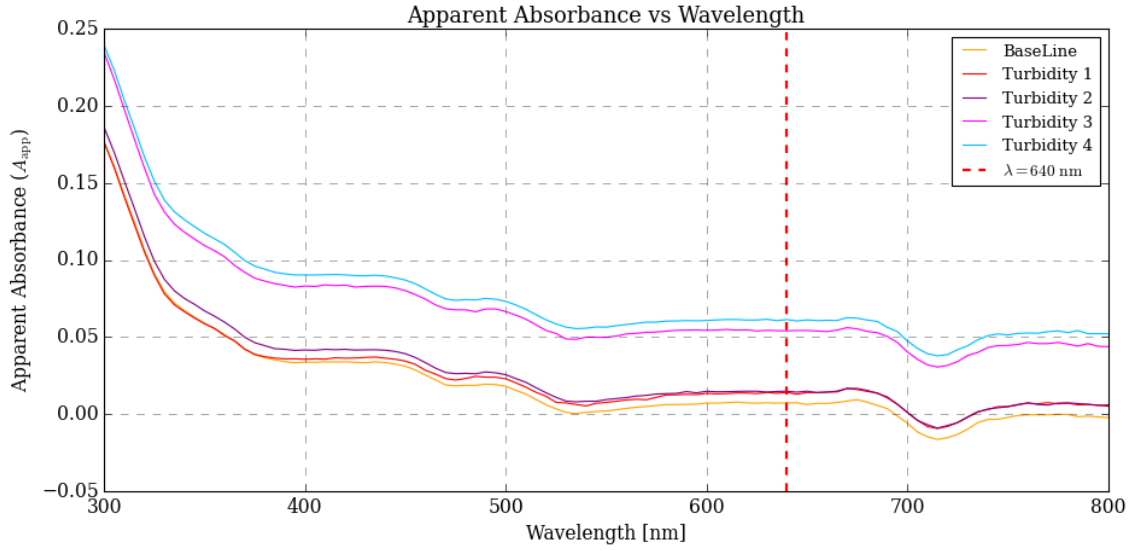


Figure 3.14: Graph of Apparent Absorbance Vs Wavelength

Sample	l [m]	A_{app}	c_{app}	R	μ_a	μ_s
Baseline	0.01	0.00729	1.678776	25	0.064568	1.614208
Sample 1	0.01	0.014091	3.245085	25	0.124811	3.120274
Sample 2	0.01	0.014849	3.419752	25	0.131529	3.288223
Sample 3	0.01	0.054054	12.44875	25	0.478798	11.96995
Sample 4	0.01	0.061359	14.13089	25	0.543496	13.5874

Table 3.4: Apparent absorbance and derived optical coefficients for all samples at 640 [nm] with a ratio of 25

3.4 Visual Results from Laboratory Testing

During each experiment, a custom script was used to extract individual frames from the recorded video sequences (see Appendix F). From these, the highest-quality frame per experiment was manually selected. Despite the fixed camera position, suspended particles present in the water were visible in several of the images due to the turbidity of the medium.

Once the images were selected for testing the neural network’s performance, manual semantic segmentation was performed using the free software Photopea [46]. Figures 3.15 to 3.23 illustrate the selected images alongside their corresponding manually annotated segmentation masks. The RGB color code assigned to each class remains consistent across the project, including both the synthetic data generated in Blender and the input provided to the CNN via the Matlab processing script.



Figure 3.15: Baseline of experimental images

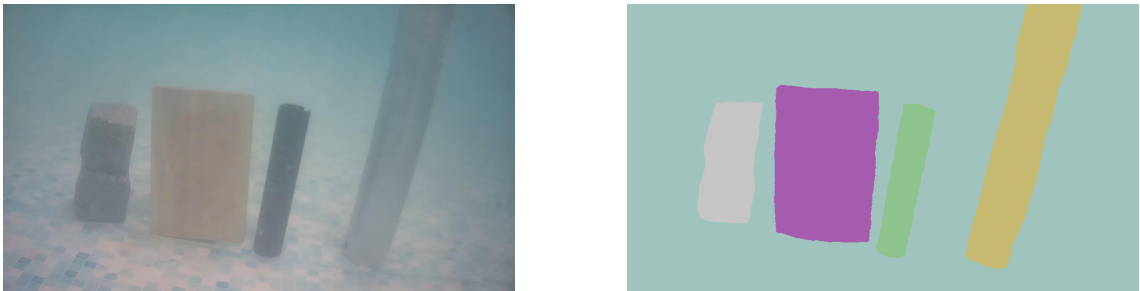


Figure 3.16: Position 1 - Turbidity 1

3. Experiment setup



Figure 3.17: Position 1 and Turbidity 2

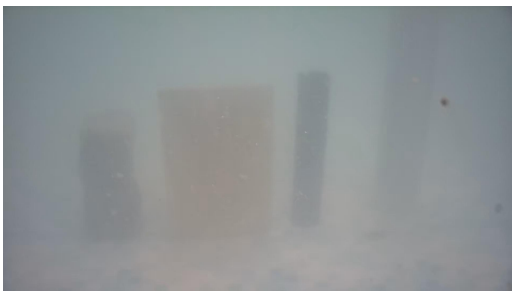


Figure 3.18: Position 1 and Turbidity 3

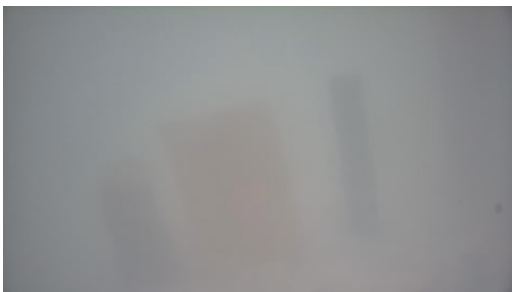


Figure 3.19: Position 1 and Turbidity 4

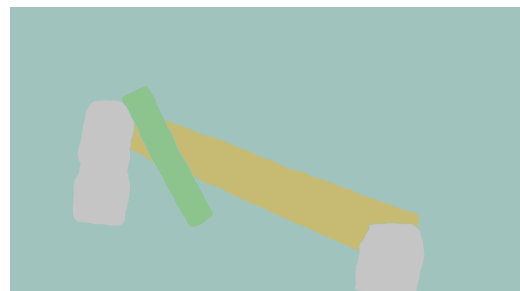


Figure 3.20: Position 2 - Turbidity 1

3. Experiment setup

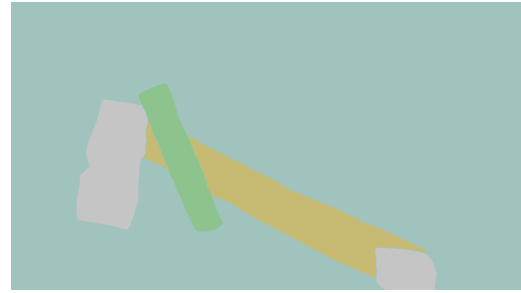
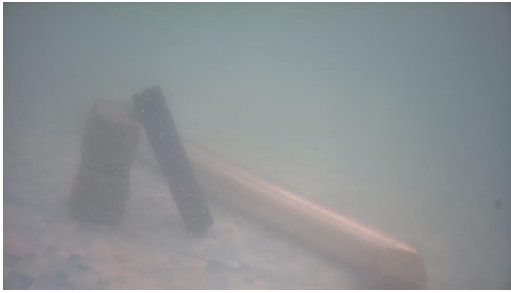


Figure 3.21: Position 2 - Turbidity 2

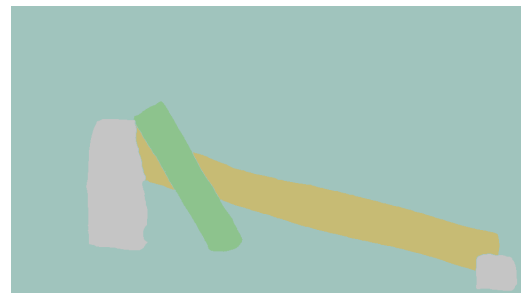
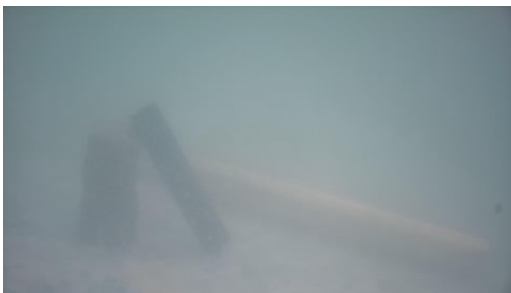


Figure 3.22: Position 2 and Turbidity 3

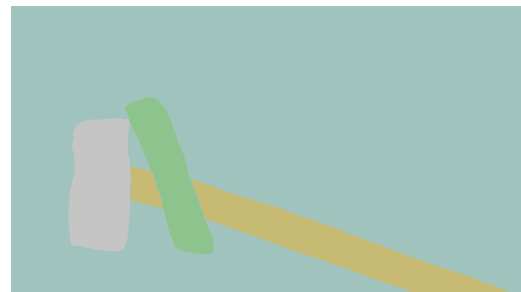


Figure 3.23: Position 2 and Turbidity 4

4 Synthetic Images

Synthetic data refers to information generated artificially. For this project the synthetic data are the generated virtual pictures. The employed tool is Blender [47], which is open-source and offers a broad range of customizable options, online repositories, and extensive documentation. Additionally, it's free and supports Python integration for custom modifications.

The synthetic data generated in this project can be categorized into two distinct types. First, photorealistic images resembling those captured by the experimental camera (see Section 3.4); and second, corresponding segmentation masks derived from those images.

This mask is an ordered composition of the classes to identify in the image, where the RGB values of each pixel are related to a particular class. Therefore the variety of colors in the segmentation mask is very small, there are as many colors in the colloquial sense as there are classes. A visual example of a data pair generated for this project can be found in image 4.1.

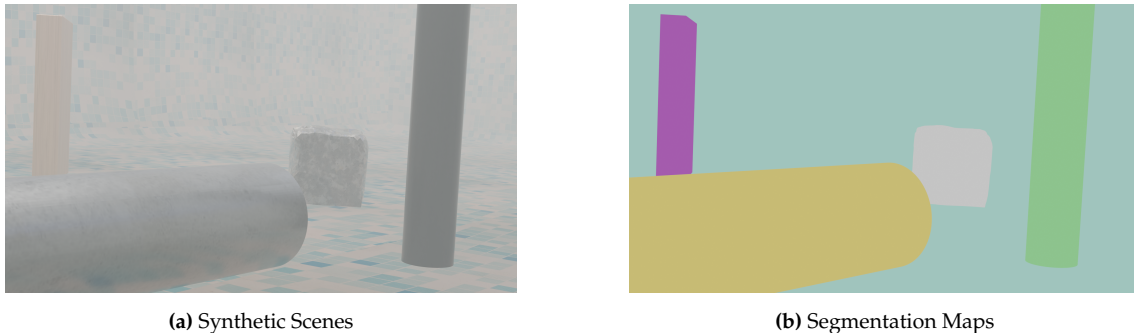


Figure 4.1: Realistic Appearance Images vs Segmentation Maps

The main advantage of synthetic data is the cost savings compared to real-world images, which would require a team of professionals and specialized equipment such as ROVs. In addition, capturing real footage offshore entails transportation logistics and planning around favorable weather conditions to ensure a safe operation. For instance, manually classifying the nine selected images from the experimentation, pixel by pixel for the five chosen classes, took an entire day of work.

Another advantage is its high variability, allowing to the simulation of very diverse states and environments, being these easily modifiable. Some manipulable elements are the material properties, the light in the captured scene, the reflections, different degradation states of the objects, etc. Being able to generate data batches with diverse environments enriches the network by providing more general features and therefore increasing prediction accuracy [48], [49].

To provide a clear overview of the process involved in generating synthetic data, the following steps were carried out:

1. The meshes of the objects were designed, including both the main study elements and their surrounding environment.
2. Appropriate materials were assigned to each object to realistically simulate surface properties.
3. The optical properties of the water, specifically light absorption and scattering, were modeled to replicate subsea conditions.
4. Scene lighting was configured to simulate real-world illumination scenarios.
5. Camera settings, render engine parameters, and other computational properties were adjusted to ensure consistent output quality.
6. A script was developed to automatically generate segmentation masks for the simulated images, storing the results in a structured and classified format.
7. The diversity of the virtual environments was increased to enhance the representativeness and robustness of the synthetic dataset.
8. The outcomes of the synthetic data generation were analyzed to iteratively improve the composition and realism of the scenes.

Throughout this chapter, the most relevant processes involved in the creation of synthetic data are described in detail.

4.1 Mesh modeling

The first step in modeling the objects was to establish a clear structure outlining the key elements to address. This included defining the modeling strategy, selecting the objects to be represented, and determining their corresponding sizes. Consequently, it was necessary to measure the dimensions of each object in advance.

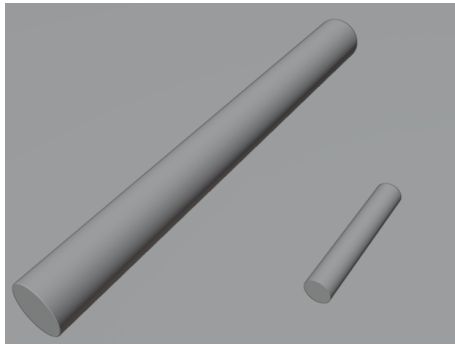
Once Blender was launched, the objects were created within the “Layout” workspace by inserting basic geometric primitives such as cubes or cylinders. These shapes were then edited to match the desired symmetry. Subsequently, materials were assigned to each object according to their specific characteristics.

4.1.1 Pipes

For the pipes, both the aluminum (referred to as the silver pipe throughout the project) and the black one, the creation process was as described below.

A cylindrical shape was selected as the base geometry using the mesh option in Object Mode, with approximate dimensions set accordingly. The number of vertices was then defined; this parameter determines the resolution of the circular faces. Since achieving a perfectly smooth curvature was not essential for the objectives of this project, a total of 128 vertices were selected.

Once the final form was completed, a bevel was applied to trim the edges; this step was performed on all objects to make the details more closely resemble real-world appearances. The bevel size and edge smoothing settings varied depending on the object. The selected measurements and an image of the mesh are shown in Figure 4.2.



(a) Smoothed mesh of pipes

Object	Element	Measure [m]
Black Pipe	Length	0.30
Silver Pipe	Length	0.95
Black Pipe	Radius	0.025
Silver Pipe	Radius	0.050

(b) Pipe dimensions

Figure 4.2: Beveled and smooth-shaded pipe meshes alongside their dimensions

4.1.2 Granite cobblestone

The workflow for the granite model followed the same steps. First, a large cube with dimensions of 0.4 [m] was added in the Layout workspace. Then, work was carried out in the Sculpting workspace, where the design Rock and Stone Brushes addon [50] was applied using imported brushes to achieve the desired surface texture. Next, in the UV Editing workspace, the vertex density was adjusted, the mesh was remeshed, and the faces were redistributed to create a more heterogeneous rock surface. This process, known as UV unwrapping, was followed by scaling the UV islands to match the dimensions of the granite pieces used. Three granite models were created, as shown in Figure 4.3.

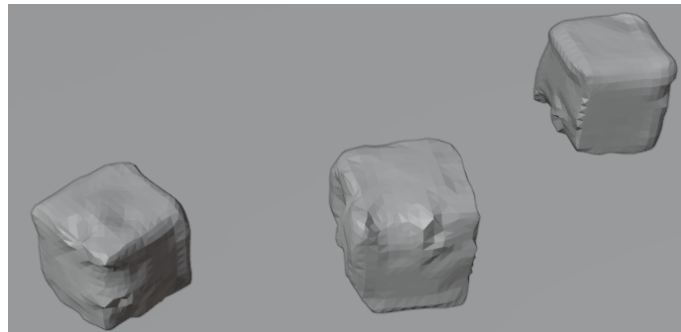


Figure 4.3: Granite Mesh

4.1.3 Wooden Piece

The wooden piece was created starting from a cubic volume. Using the scale tool, its dimensions width, height, and thickness were adjusted to match those of the real wooden object, measuring 19 [cm] \times 30 [cm] \times 5 [cm] . Figure 4.4 shows the resulting 3D model of the wooden table.

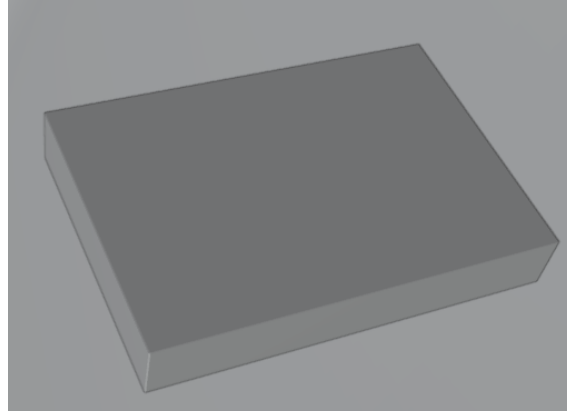


Figure 4.4: Wooden Piece Mesh

4.1.4 Pool

During the early stages of project development, attempts were made to model the shape of the pool, however due to its high complexity and number of polygons and nuances to take into account, it was decided to purchase the pool model, scale it manually in the x, y and z coordinate axis, so that its measurements would match those of the laboratory. The address of the purchased model as well as information on the created modeling artists can be found in [51]. Figure 4.5 represents the quality of the pool mesh.

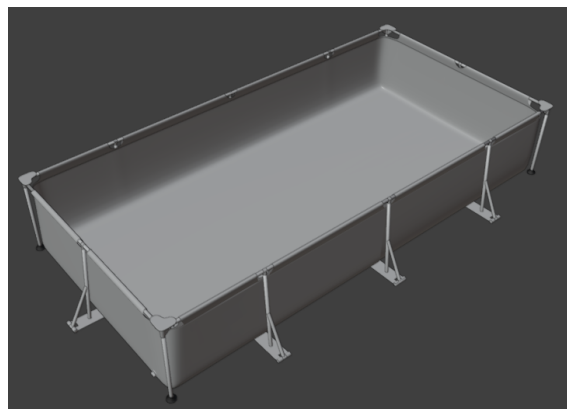


Figure 4.5: Pool Mesh

4.2 Object's Material

An essential part of the object representation process was the selection and application of appropriate materials to simulate realistic surface properties. To support this task, exter-

nal resources such as the BlenderKit Asset Library [52] were employed, offering a broad collection of user-generated assets and textures.

For elements like the pipes and the wooden box, predefined texture packs were downloaded and directly applied to the respective models. In contrast, the granite stone material was selected manually to achieve a more accurate visual representation. Additionally, the interior lining of the pool was recreated by capturing a photograph of the actual pool bottom and mapping it onto the model's surface.

In Section 4.3.1, the results obtained following the addition of textures are presented.

4.2.1 Granite Material

The texture of granite stones was searched on the Poligon website [53], the texture pack [54] was downloaded and assembled manually, and figure 4.6 contains the employed images. The blocks involved in the material setup are detailed in the list below. Figure 4.7 shows the block specifically applied to simulate the granite surface.

1. Block Image Texture (Base Color) provides the image of what the material looks like in a flat form and feeds the Principled BSDF.
2. Image Texture (Roughness) loads the roughness map. Overlays a grayscale image, so the diffusion of light on the object is different in each area.
3. Image Texture (Normal) and Normal Map here it is loaded the RGB normal map, which converts color vectors into altered surface normals. These modified normals simulate fine details such as scratches without changing the mesh geometry.
4. Image Texture (Height) + Displacement the height map generates a real geometry displacement by applying a grayscale image and a push or pull command based on the texture image.
5. Principled BSDF combines all the properties in one shader to provide photorealistic appearance.
6. Material output apply final surface shading and geometry displacement.



Figure 4.6: Granite Textures: Base Color - Height - Roughness - Normal

4. Synthetic Images

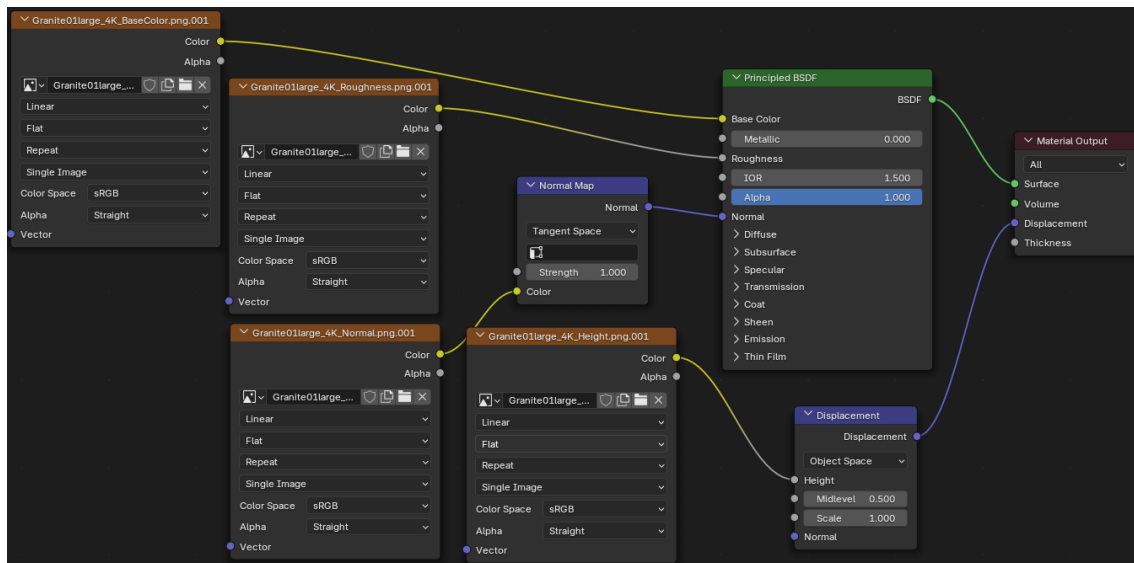


Figure 4.7: Granite Material Architecture

4.2.2 Aluminum Pipe Material

The material used has been downloaded from Blender Kit, This metallic material has been chosen for its great similarity to the real aluminum pipe. The following list highlights the components. The structure is quite similar to the previous case, but has an added block Texture Coordinate + Mapping that provides the coordinates and applies location/rotation/scale adjustments to control texture placement. Figures 4.8 and 4.9 detail the aluminum textures.

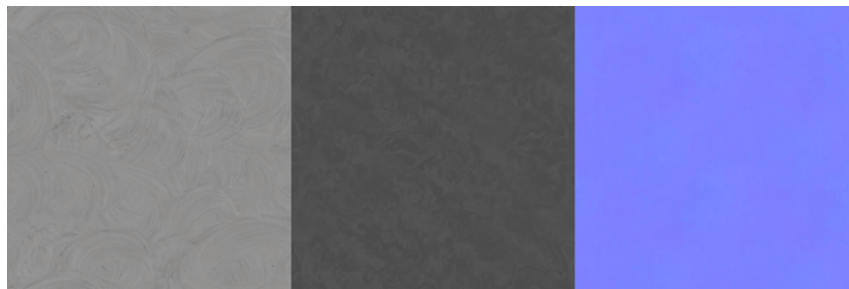


Figure 4.8: Aluminum Textures: Base Color - Roughness - Normal

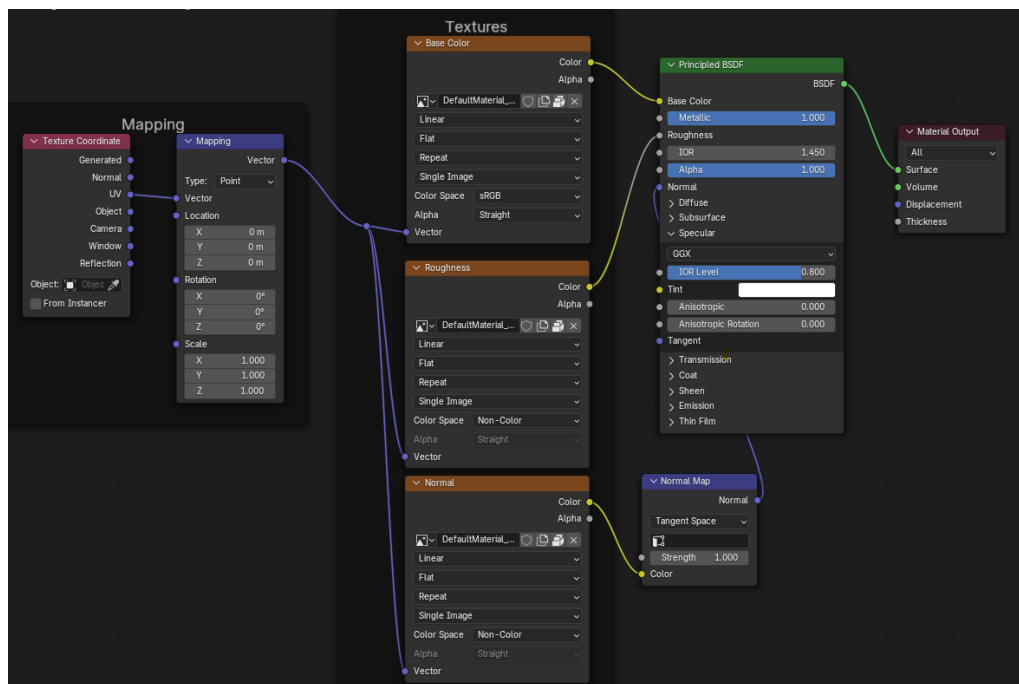


Figure 4.9: Silver Aluminum Material Architecture

4.2.3 Black Pipe Material

The following material simulates a black, opaque pipe with subtle metallic highlights. Only the Principled BSDF shader has been used, the RGB values are 0.13 in each channel, and the rest of the properties are visible in the image 4.10.

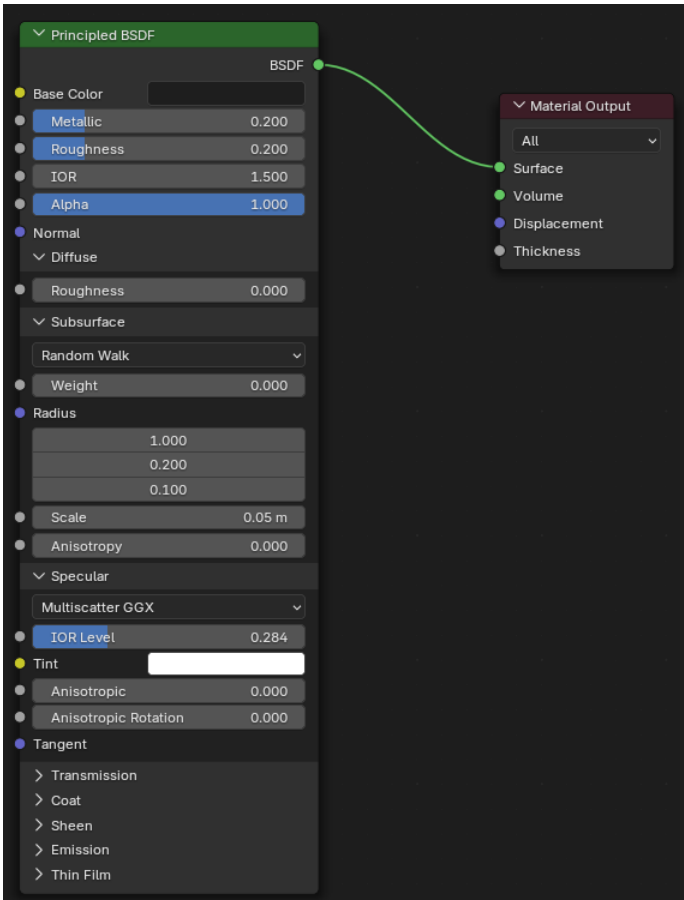


Figure 4.10: Black Pipe Material Architecture

4.2.4 Wooden Piece Material

A pine wood material was chosen for its natural coloration, affordability, and common usage. The architecture is identical to case Aluminum Pipe Material 4.2.2 where the input textures are modified to simulate wood. Figures 4.11 and 4.12 encapsulates the texture and architecture.

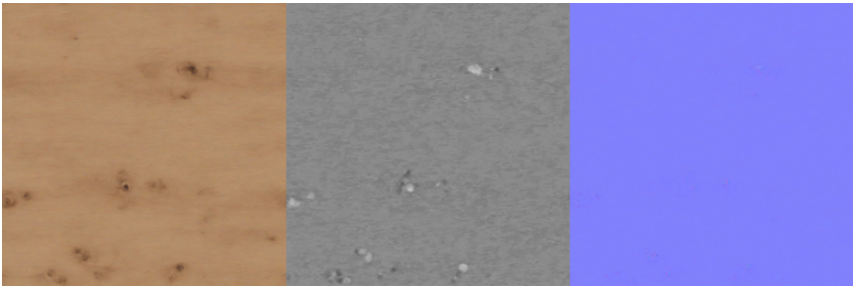


Figure 4.11: Wooden Piece: Base Color - Roughness - Normal

4. Synthetic Images

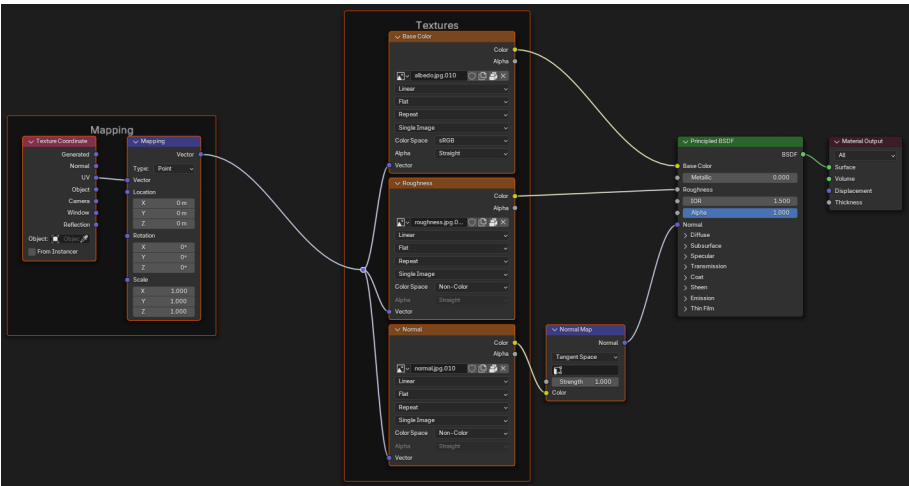


Figure 4.12: Wooden Piece Material Architecture

4.2.5 Pool Background Material

The pool object was a prebuilt asset with its own texture pack, The texture pack is not detailed here except for the replaced background texture, which is a picture of the bottom of the pool in the flow lab 4.13. For this reason, only the use of the image and the shader principle BSDF is shown 4.13.

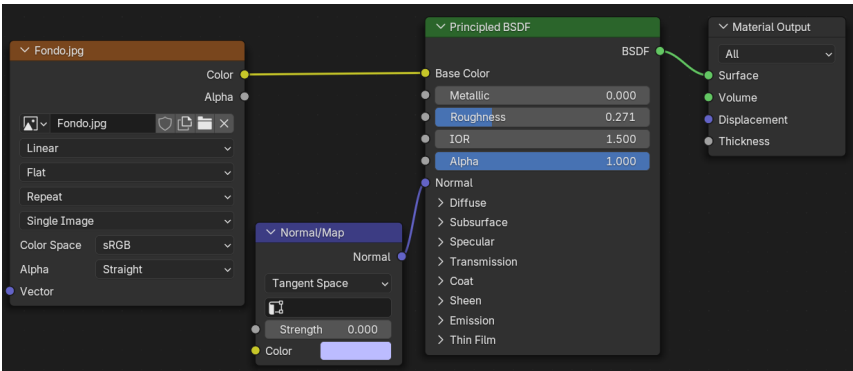


Figure 4.13: Pool's Background Material

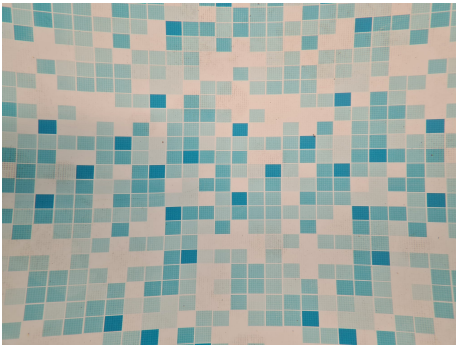


Figure 4.14: Bottom part of the laboratory's pool

4.3 Light Modeling in Blender

First, the fluorescent tube lights used in the laboratory are identified [24]. Then, the same scene is recreated with the configuration 4.15. In Blender, light power values are not directly equivalent to real-world wattage. For instance, simulating an 80 [W] fluorescent tube (approximately 6150 lumens) often requires setting the power parameter to a significantly higher value in Blender [55].

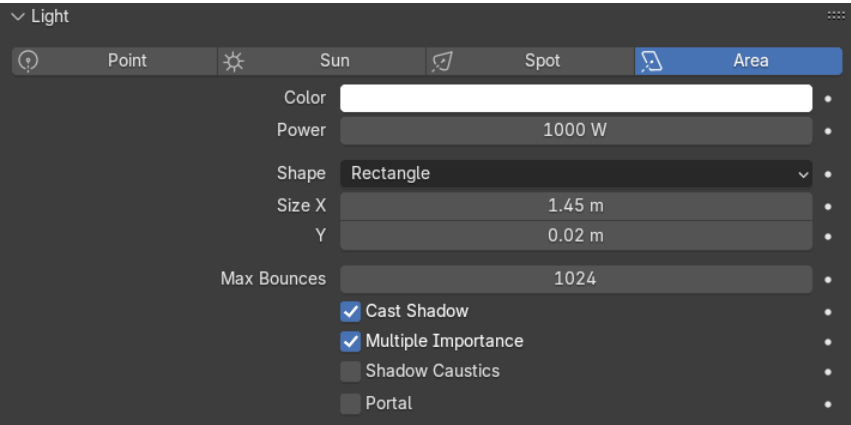


Figure 4.15: Light Properties

Parameter	Values
Light Type	Area
Shape	Rectangle
Size (X)	1.45 m
Size (Y)	0.02 m
Color	#fffffa
Power	1000 W
Multiple Importance	Enabled
Cast Shadow	Enabled

Table 4.1: Settings in Blender to simulate an 80W T5 fluorescent tube

4.3.1 Visual Comparison of Model Stages

Once the mesh modeling and the material have been explained, a visual comparison is presented to show the simulated model’s progression. This little section highlights the transition from mesh only to mesh with textures. Figure 4.16 serves as a point of comparison.

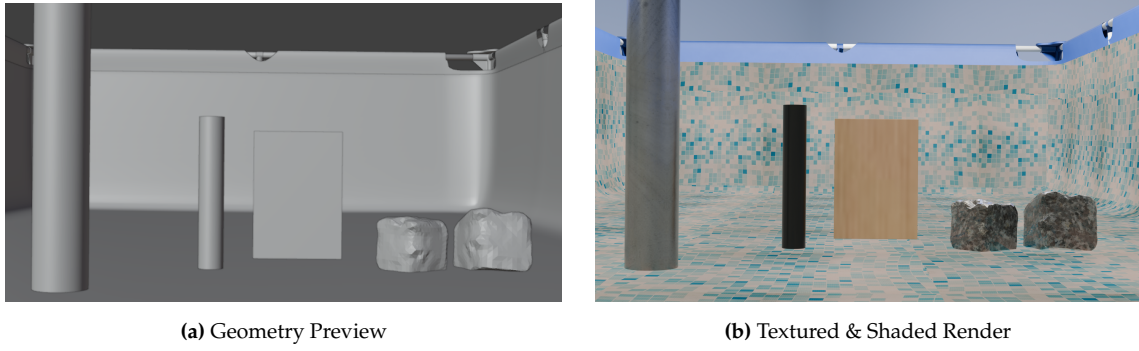


Figure 4.16: Mesh vs. Textured Visual Comparison

4.4 Water Volume Properties

The water volume was simulated using the structure described in [56]. The correlation between the light scattering and absorption is described in chapter 3.3, an image of the architecture is found in figure 4.18. In this subsection, the effect of kaoling mixed with water is represented.

- Absorption coefficient μ_a [m^{-1}]: controls light attenuation through the volume.
- Scattering coefficient μ_s [m^{-1}]: controls light diffusion by suspended particles.
- Anisotropy factor g [-]: sets the preferred scattering direction.

The initial absolute attenuation or apparent absorbance values are taken for the 640 [nm] wavelength, measured from water samples from the four tests performed. The contributions due to the absorption coefficient and the scattering coefficient are calculated and manually added to the material.

Turbidity	A_{app}	μ_a	μ_s
Reference	0.00728	0.06456	1.61420
Low	0.01409	0.12481	3.12027
Medium	0.01484	0.1315	3.28822
Moderate	0.05405	0.47879	11.96994
High	0.06135	0.54349	13.58739

Table 4.2: Turbidity grades with corresponding attenuation, absorption, and scattering coefficients.

However, when simulating the measured values, these barely show a contrast as marked as in reality, i.e., the synthetic images obtained by using the actual measured values showed less turbidity than the real images. Therefore it was decided to manually tune these values as well as the Absorption and Scattering density, so that the executed renders are similar to the captured experimental images. The absorption color is identical for

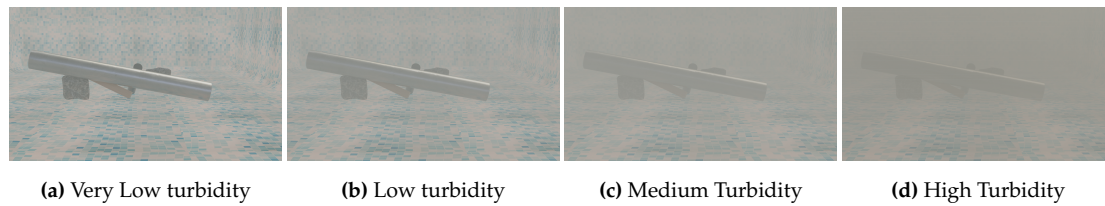


Figure 4.19: Turbidity Levels

4.5 Render Properties

In Blender, rendering refers to the process of transforming a three-dimensional scene—including geometry, lighting, camera settings, and materials—into a two-dimensional image or animation. This involves simulating how light interacts with surfaces to produce realistic visual effects such as color, shadows, and reflections.

During the initial stages of the project, the EEVEE render engine was employed due to its faster rendering capabilities, which facilitated rapid testing and scene adjustment. However, EEVEE is a real-time, rasterization-based engine and does not offer the level of physical accuracy required for high-fidelity visual output. As a result, several visual artifacts were observed, particularly in the rendering of reflective and metallic surfaces, as well as in the fine details of the granite cobblestones (see Figure 4.20).

To address these limitations, the Cycles engine was adopted for the final image generation. Cycles is a physically-based, path-tracing renderer that simulates the behavior of individual light rays as they interact with the scene. Although more computationally demanding, this engine provides superior results in terms of lighting realism, shadow accuracy, and material representation—features that are essential when generating synthetic images intended to mimic real-world underwater environments.

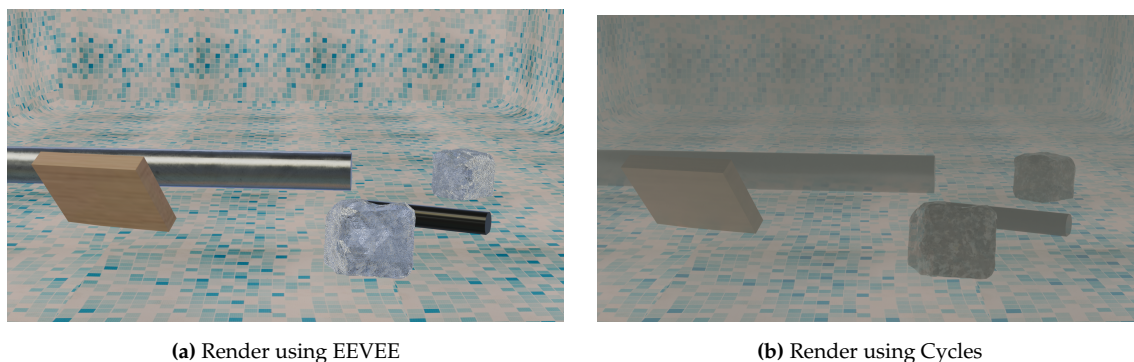


Figure 4.20: Comparison between rendering engines for a high turbidity condition

To accelerate the image generation process, several render optimization settings were applied in Blender. First, the rendering device was switched from the CPU to the GPU, as modern GPUs are highly parallelized and significantly outperform CPUs in rendering

tasks, especially when using path-tracing algorithms.

Second, the noise threshold parameter was enabled and set to 0.1. This value controls the adaptive sampling behavior in Cycles: during rendering, Blender estimates the noise level in different parts of the image, and if the noise falls below the threshold (in this case, 0.1), it stops calculating additional light samples for that pixel. This prevents unnecessary computation in well-converged regions, helping to reduce render times. Conversely, if the noise remains high, the renderer may increase the sample count up to a maximum limit (1024) to improve image quality.

Third, a maximum render time of 120 seconds per image was imposed to avoid excessive computation in complex scenes. This time cap ensures a balance between visual quality and dataset generation efficiency.

Overall, these settings were chosen to optimize the trade-off between speed and quality, enabling the generation of high-fidelity synthetic images in a reasonable timeframe.

The camera resolution was chosen based on the size of the images extracted from the real camera and the field of view, in order to achieve similar results when comparing the Blender renders with the experimental photographs. The image resolution was set to 1920×1080 pixels, which was configured using the output resolution settings.

The virtual camera settings were adjusted after selecting the camera object. In the camera properties panel, the lens type was set to perspective, and the focal length was adjusted to 45.22 [mm] to match the real camera. The sensor camera size was kept at 28 [mm].

To assess the quality of the synthetic images presented in Figure 4.19, a visual comparison was conducted against the real images obtained during the experimental phase (see Section 3.4). The evaluation relied on a subjective method in which several individuals were asked to distinguish between real and synthetic images. When respondents encountered difficulty in differentiating them and provided incorrect classifications, the synthetic renderings were deemed sufficiently realistic for the purposes of this study.

4.6 Camera Pose Sampling

To automatically create the renders, a distribution of camera positions and orientations was added to the code, always pointing at the central area of the pool, (since the elements to be captured were located in the central area) from different heights and sides, to obtain different viewpoints of the objects being studied. The positions used are shown in the following image.

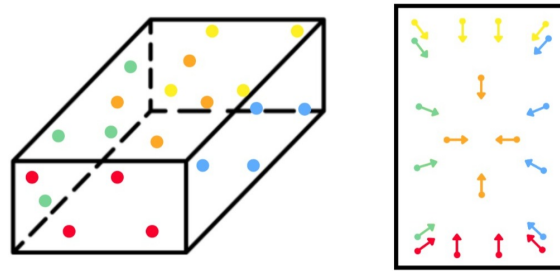


Figure 4.21: Camera distribution over a rendered scene

4.7 Synthetic Image Pair Creation

The training process selected for the CNN is supervised, where it is provided with an object classification with pixel by pixel information. The network is trained in pairs; the process of creating unclassified synthetic images has been explained in the previous sections. To create the mask, a script has been developed in Blender, in the Python language; this script will be detailed in chapter 5.1.1. This section indicates its operation at a high level.

The mask is created by selecting a pass index for each object. This index is a label used to identify each object. A new material is then generated. This material is applied to all objects (pipes, granite, wood, and pool) and assigns them specific RGB values by index, so each element has a different, known color. The image is then rendered, creating the raw segmentation mask.

The constructed material provides a known color for each object. Its construction is based on blocks; the colors used do not follow any specific criteria. The architecture is explained in the following list and can be visualized in figure 4.22. The images obtained will be processed before being fed to the neural network.

1. Object identifier: assigns a unique integer to each object.
2. The node Value: indicates how many objects are expected to be colored in the image.
3. The Divide node: normalizes the object index to a $[0-1]$ range, which is exactly what the Color Ramp expects at its Fac input.
4. Color Ramp: takes the normalized value $[0-1]$ and converts it to a color according to the ramp stops.
5. Emission shader: uses the ramp's output color to emit light of that hue.
6. Material Output: connects the Emission shader to the Surface socket so the object appears self-lit in that color.

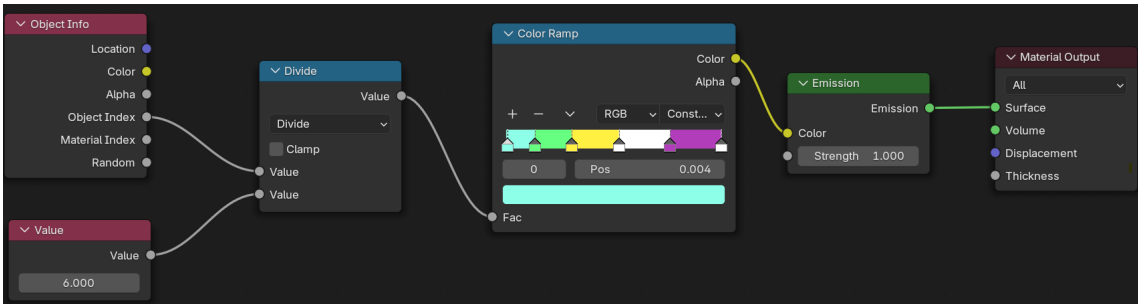


Figure 4.22: Segmentation Mask Architecture

4.8 Other simulated environments

Additionally, to train the broader neural network model, feeding it with greater image variability to enrich the predictions. It has been decided to modify the water color as well as the light intensities to generate more conditions. Some examples of these images are found in Figure 4.23.

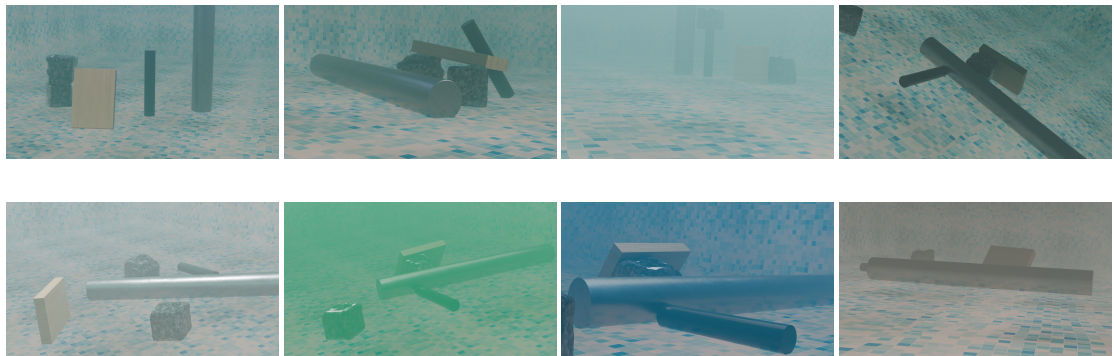


Figure 4.23: Collection of additional Simulated Underwater Conditions

5 Convolutional Neural Network in matlab

Neural networks can be developed using various programming languages and specialized frameworks. Commonly used languages include C++, R, Java, and particularly Python, which is widely adopted due to its simplicity and extensive support libraries. Several frameworks are available to facilitate neural network development, such as TensorFlow, PyTorch, Keras, CNTK, and Matlab.

For this project, Matlab was selected as the development environment due to prior familiarity and its integrated toolboxes. Specifically, the Deep Learning Toolbox and the Deep Network Designer were employed, in combination with official tutorials and example-based documentation [57]–[59].

This chapter outlines the methodology used for preprocessing the dataset, the training process of the neural networks, and the implementation of the prediction pipeline to generate semantic segmentation masks, along with other relevant elements.

5.1 Image preprocessing

This section covers all the scripts used to train the network. Before feeding the CNN, each synthetic image and real photograph had to undergo a uniform preprocessing process. Some examples are: Resizing and Normalization.

5.1.1 Creation of the segmentation mask in blender

As explained in Chapter 4.7, Blender not only produced the original images, but also generated a copy of them by applying a segmentation mask. This subsection explains the code used step by step. Appendix B contains the full code, which is broken down and explained section by section.

Libraries and Main Folder

This script defines the libraries employed and selects the output folder of the entire code.

```
import bpy          # Blender's Python API for automation
import os           # File-system operations: paths and directory creation
import math         # Math functions, e.g. degree-to-radian conversion
# Main output folder for renders
output_dir = r"C:\Users\mikio\Desktop\Pool"
```


Scene Objects & Materials Setup

To select the objects to be displayed in the render and links them to the materials to be used, both for the original and segmented images. Therefore, the name used for each object must be unique and assigned in the framework.

```
# Objects to render
paisaje_objs = [
    bpy.data.objects["BlackCilinder"],
    bpy.data.objects["SilverCilinder"],
    ...,
    bpy.data.objects["Water"]
]
# Their original materials, with paisaje_objs
mats_paisaje = [
    bpy.data.materials["mat11"],
    bpy.data.materials["Frozen white metal"],
    ...,
    bpy.data.materials["WaterProp"]
]
# Segmentation material (a flat, index-colored shader)
matsegmentacion = bpy.data.materials["matsegmentacion"]
```

Camera Position Definitions

The number of camera locations is indicated, both the orientation and the position, the camera always aims to the center of the pool, where the objects are located. Section 4.6 introduces these positions.

```
# A list of (location, rotation) tuples for the camera
cam_positions = [
    ([2.1876, -0.139, -0.13949], [math.radians(85.15), ...]),
    ...,
    ([0.2708, 0.48936, 0.58282], [math.radians(161.41), ...])
]
```

Object Placement

The object configuration process is automated to save time, thus obtaining more renders in the same amount of time spent.

```
composiciones_objetos = [
    { "SilverCilinder": {"loc": ..., "rot": (...)} ,
    ... }, { ... }, { ... } ]
```

Loop objects positioning

Iterates setting each object's position and rotation.

```
def aplicar_composicion(compo_dict):
    for name, vals in compo_dict.items():
        obj = bpy.data.objects[name]
        obj.location = vals["loc"]
        obj.rotation_euler = vals["rot"]
```

Main render function

Creates two folders in the output directory, one for the segmentation mask renders and other for the raw images, checks for the last numerical number in the folders and continues with the upcoming one. Calls for the camera and objects positioning, later starts the renders firstly for the segmentation mask and secondly for the raw images.

```
def render_segmentacion_y_raw(output_dir, paisaje_objs,
    mats_paisaje, matsegmentacion, cam_positions):
    # Create output folders
    mascara_dir = os.path.join(output_dir, "mascara_segmentacion")
    raw_dir      = os.path.join(output_dir, "raw_imagenes")
    os.makedirs(mascara_dir, exist_ok=True)
    os.makedirs(raw_dir, exist_ok=True)

    scene = bpy.context.scene
    cam    = scene.camera

    # get next free index based on existing PNGs
    def get_next_index(folder):
        existing = [f for f in os.listdir(folder) if f.endswith(".png")]
        if not existing:
            return 1
        nums = [int(f.split(".")[0]) for f in existing if f.split(".")[0].isdigit()]
        return max(nums) + 1 if nums else 1

    idx = get_next_index(mascara_dir)

    # Loop over each camera pose
    for pos, rot in cam_positions:
        frame_name = f"{idx:04d}"

        # Move & rotate camera
        cam.location      = pos
        cam.rotation_euler = rot
```

```
# 1) Segmentation mask render
for obj in paisaje_objs:
    obj.active_material = matsegmentacion
    scene.render.filepath = os.path.join(mascara_dir, frame_name + ".png")
    bpy.ops.render.render(write_still=True)

# 2) Raw render with original materials
for obj, mat in zip(paisaje_objs, mats_paisaje):
    obj.active_material = mat
    scene.render.filepath = os.path.join(raw_dir, frame_name + ".png")
    bpy.ops.render.render(write_still=True)

idx += 1 # Increment filename index
```

5.1.2 Image grouping and classification

One of the prior steps before feeding the CNN was to merge all the data into one folder for each specific purpose. This script was developed because two different computers were employed to render the data set images.

```
% Paths to add the images (output directories)
dest_mask_dir = 'C:\Users\mikio\Desktop\Pool\mascara_segmentacion';
dest_raw_dir  = 'C:\Users\mikio\Desktop\Pool\raw_imagenes';

% Paths of folders to take the data (input directories)
src_mask_dir = 'C:\Users\mikio\Desktop\Turbidity2&3\mascara_segmentacion';
src_raw_dir  = 'C:\Users\mikio\Desktop\Turbidity2&3\raw_imagenes';

% Search the last number in the output folder
mask_files = dir(fullfile(dest_mask_dir, '*.png'));
raw_files  = dir(fullfile(dest_raw_dir, '*.png'));

% Obtain the last figure (asuming names follow the structure 0001.png)
last_idx = 0;
if ~isempty(mask_files)
    nums = arrayfun(@(f) str2double(f.name(1:4)), mask_files);
    last_idx = max(nums);
end

% list the source files
src_mask_files = dir(fullfile(src_mask_dir, '*.png'));
src_raw_files  = dir(fullfile(src_raw_dir, '*.png'));
```

```
% Order both
[~, idx_mask] = sort({src_mask_files.name});
src_mask_files = src_mask_files(idx_mask);
[~, idx_raw] = sort({src_raw_files.name});
src_raw_files = src_raw_files(idx_raw);

% Copy and rename keeping the parity relation
for i = 1:min(length(src_mask_files), length(src_raw_files))
    % Nuevo índice global
    new_idx = last_idx + i;
    new_name = sprintf('%04d.png', new_idx);

    % Mask copy
    copyfile(fullfile(src_mask_dir, src_mask_files(i).name), ...
        fullfile(dest_mask_dir, new_name));

    % Raw copy
    copyfile(fullfile(src_raw_dir, src_raw_files(i).name), ...
        fullfile(dest_raw_dir, new_name));
end

disp('Combinación y renumeración completadas.');
```

5.1.3 Normalization and resizing of the segmentation Mask

During the image classification process before feeding the network, it became apparent that the images selected in the segmentation mask did not contain unique RGB values for each class. To address this problem, code was written to look for similar colors, allowing for a difference of 4 units in the scale of each RGB channel (0-255), and replace them with the colors assigned in Blender. The image dimensions were also resized from 1920x1080 to 960x540 to reduce computational costs. A similar script was created for the raw photos, but it won't be shown. Only the segmentation values are shown in this section.

```
inputFolder = 'C:\Users\mikio\Desktop\Turbidity2&3\CamVid\mascara_segmentacion';
outputFolder = 'C:\Users\mikio\Desktop\Turbidity2&3\CamVid\labels';
newSize = [540, 960]; % [alto, ancho]
tolerance = 4;

% RGB values for each class
classColors = [
    160 196 189; % Pool's background
    141 195 141; % Black cilinder
```

5. Convolutional Neural Network in matlab

```
199 187 116; % Silver cilinder
165  92 174; % Wooden piece
197 197 197  % Granite cobblestones
];

% Create output directore
if ~exist(outputFolder, 'dir')
    mkdir(outputFolder);
end

% Read PNG files
files = dir(fullfile(inputFolder, '*.png'));

for k = 1:length(files)
    % Read and resize the pictures
    img = imread(fullfile(inputFolder, files(k).name));
    img = imresize(img, newSize);

    % Initialize corrected image with background color (class 1)
    correctedImg = uint8(ones(size(img)) .* reshape(classColors(1, :), 1, 1, 3));

    % For each class, correct pixels within tolerance
    for i = 1:size(classColors, 1)
        targetColor = classColors(i, :);

        % Create mask for pixels similar to targetColor
        mask = all(abs(double(img) - reshape(targetColor, 1, 1, 3)) <= tolerance, 3);

        % Apply the class color to matching pixels
        for c = 1:3
            channel = correctedImg(:,:,c);
            channel(mask) = targetColor(c);
            correctedImg(:,:,c) = channel;
        end
    end
end

% Save corected image
imwrite(correctedImg, fullfile(outputFolder, files(k).name));
end
```

5.2 Trainig of ResNet-18

For the training of the network, different strategies and an iterative trial and error method was followed, until some reasonable results were obtained. Some inspirational examples include a Matlab exercise where a ResNet-18 network was trained for a single channel and one class [59].

The code will be divided into smaller and more understandable parts. The original code is found in appendix C. This code is presented for training the network under the lowest turbidity condition. Scripts with identical structures have been developed, but they adapt differently to the different cases analyzed. However, to use a logical comparison criterion, the training options have not been altered from one case to another.

Paths, classes and labelIDS

In this part of the code, the routes where the original and classified images are stored are defined, the classes with their colors in RGB are defined, and the datastore variable that stores all this information is generated.

```
%Here we set the path to images and label folders
rutaImagenes = 'C:\Users\mikio\Desktop\EntrenamientoT1\CamVid\images';
rutaEtiquetas = 'C:\Users\mikio\Desktop\EntrenamientoT1\CamVid\labels';

% class names
classes =
    [
        "Pool_background"
        "Black_Pipe"
        "Silver_Pipe"
        "Wooden_Box"
        "Granite"
    ];

% Define the RGB colors associated in order
labelIDS =
    [
        160 196 189; % Pool's background
        141 195 141; % Black cilinder
        199 187 116; % Silver Cilinder
        165  92 174; % Wooden box
        197 197 197  % Granite
    ];

% Create the datastore
```

```
imds = imageDatastore(rutaImagenes);  
pxds = pixelLabelDatastore(rutaEtiquetas, classes, labelIDs);
```

Data Partitioning

This section shows how the dataset was divided and the percentages used to train and validate the network.

```
% Split the data into 80% for training and 20% for validation
```

```
% Total number of images  
numFiles = numel(imds.Files);
```

```
% Create a random index  
randIdx = randperm(numFiles);
```

```
% Percentage for training  
trainRatio = 0.8;  
numTrain = round(trainRatio * numFiles);
```

```
% Indexes for training and validation  
trainIdx = randIdx(1:numTrain);  
valIdx = randIdx(numTrain+1:end);
```

```
% Split the image datasore  
imdsTrain = subset(imds, trainIdx);  
imdsVal = subset(imds, valIdx);
```

```
% Split the label datasore  
pxdsTrain = subset(pxds, trainIdx);  
pxdsVal = subset(pxds, valIdx);
```

Once the raw and segmentation folders have been divided between training and validation, they are separated into only training and only validation, and the type of network to be used is also indicated.

```
% Combine the training datastore  
trainingData = combine(imdsTrain, pxdsTrain);
```

```
% Combine validation datastore  
validationData = combine(imdsVal, pxdsVal);
```

```
%Define the size of the images and the number of channels
imageSize = [540, 960, 3]; % height, width and channels
numClasses = 5;
network = 'resnet18'; % Network employed
```

5.2.1 Training options

Standard training options and layer graph for a DeepLab v3+ semantic segmentation network are specified in the code below, and the network training begins by selecting the training data, options, and layer graph. Finally the network is inspected and stored.

```
options = trainingOptions('sgdm', ...
    'InitialLearnRate',1e-4, ...
    'MaxEpochs',30, ...
    'MiniBatchSize',6, ...
    'Shuffle','every-epoch', ...
    'VerboseFrequency',10, ...
    'Plots','training-progress', ...
    'ExecutionEnvironment','auto');

lgraph = deeplabv3plusLayers([540 960 3],5,'resnet18');
[net, info] = trainNetwork(trainingData, lgraph, options);

class(net)

save("TrainedNetworkT1.mat","net","info");
```

5.3 Segmentation mask via DeepLab v3+

The methodology followed in this project was inspired by the practical Matlab example [57], developed in collaboration with the University of Cambridge. The script was adapted to meet the specific requirements and conditions of this study. This section provides a detailed step-by-step explanation of the modified version. The complete, unabridged code can be found in Appendix D.

Class & Colormap Definitions

Defines the list of semantic classes, their corresponding integer identifiers, and the RGB category color map used during training and visualization.

```
function classes = getClassNames()
    classes =
    [
        "PoolBackground"
        "BlackCilinder"
```



```
        "SilverCilinder"  
        "Wooden Piece"  
        "Granite"  
    ];  
end  
  
function labelIDs = camvidPixelLabelIDs()  
    labelIDs =  
    {  
        [160 196 189], % PoolBackground  
        [141 195 141], % BlackCilinder  
        [199 187 116], % SilverCilinder  
        [165  92 174], % Wooden Piece  
        [197 197 197]  % Granite  
    };  
end  
  
function cmap = camvidColorMap()  
    cmap =  
    [160 196 189;  
    141 195 141;  
    199 187 116;  
    165  92 174;  
    197 197 197] / 255;  
end
```

Data Loading & Visualization

The code snippet below performs the loading of synthetic raw images along with their corresponding segmentation masks. It then displays a sample image overlaid with its ground truth mask, and concludes by calculating the number of pixels per class and plotting their frequency distribution.

```
classes = getClassNames();  
labelIDs = camvidPixelLabelIDs();  
cmap      = camvidColorMap();  
  
imgDir    = fullfile(outputFolder, "images");  
imds      = imageDatastore(imgDir);  
  
pxds      = pixelLabelDatastore(...  
    fullfile(outputFolder, "labels"), ...  
    classes, labelIDs);
```

```
% Example display
I = histeq(readimage(imds,25));
C = readimage(pxds,25);
B = labeloverlay(I, C, ColorMap=cmap);
imshow(B); pixelLabelColorbar(cmap,classes);

% Pixel counts & bar chart
tbl = countEachLabel(pxds);
frequency = tbl.PixelCount / sum(tbl.PixelCount);
bar(1:numel(classes), frequency);
xticklabels(tbl.Name); xtickangle(45);
ylabel("Frequency");
```

Train/Val/Test Split & Augmentation

The algorithm below performs the following operation, the database is divided into training, validation, and testing. It generates the variable datastores where the sets for each training are grouped and the images are slightly modified by applying random reflection and translation to augment the training set. Only the main structure is preserved, to view the rest of the code go to appendix D.

```
[imdsTrain, imdsVal, imdsTest, pxdsTrain, pxdsVal, pxdsTest] = ...
partitionCamVidData(imds, pxds);

dsTrain = combine(imdsTrain, pxdsTrain);
dsVal    = combine(imdsVal,   pxdsVal);

% Augmentation settings
xTrans = [-10 10]; yTrans = [-10 10];
dsTrain = transform(dsTrain, @(data)augmentImageAndLabel(data,xTrans,yTrans));
```

Network Creation & Class Weighting

The DeepLab v3+ network architecture is constructed using ResNet-18 as its backbone. Additionally, the class imbalance present in the dataset is analyzed. To address this issue, class weights are computed and incorporated into the custom loss function during training, ensuring that the network learns in a balanced way and does not become biased toward the most frequently occurring classes.

```
imageSize = [960 540 3];
numClasses = numel(classes);
network    = deeplabv3plus(imageSize, numClasses, "resnet18");
```

```
% Compute class weights
tbl = countEachLabel(pxds);
imageFreq = tbl.PixelCount ./ tbl.ImagePixelCount;
classWeights = median(imageFreq) ./ imageFreq;
```

Training Options & Execution

The script presented here is responsible for configuring the training options and executing the training process if required.

```
options = trainingOptions("sgdm", ...
    LearnRateSchedule="piecewise", ...
    LearnRateDropPeriod=6, ...
    LearnRateDropFactor=0.1, ...
    Momentum=0.9, ...
    InitialLearnRate=1e-2, ...
    L2Regularization=0.005, ...
    ValidationData=dsVal, ...
    MaxEpochs=18, ...
    MiniBatchSize=4, ...
    Shuffle="every-epoch", ...
    CheckpointPath=tempdir, ...
    ValidationPatience=4);

doTraining = false; % set to true for training
if doTraining
    [net, info] = trainNetwork(dsTrain, network, ...@(Y,T) modelLoss(Y,T,classWeights), ...
        options);
end
```

Evaluation on Test Image

The pretrained network is loaded. It performs semantic segmentation on one test image and overlays the prediction vs. the ground truth. Finally, it computes the Intersection over Union (IoU), which returns a bounded value between 0 and 1, indicating how close the prediction is to the ground truth.

```
load('TrainNetworkT1.mat','net');
Itest = readimage(imdsTest,1);
Cpred = semanticseg(Itest, net, Classes=classes);

% Overlay & colorbar
Bpred = labeloverlay(Itest, Cpred, Colormap=cmap, Transparency=0.4);
imshow(Bpred); pixelLabelColorbar(cmap, classes);
```

```
% Compute IoU
expected = readimage(pxdsTest,1);
iou = jaccard(Cpred, expected);
table(classes, iou)
```

Batch Evaluation & Experimental Data Test

This code computes the segmentation on all images obtained after the experimental tests in the laboratory and calculates the general metrics. It applies the trained network to these images, generating the segmentation mask prediction with the RGB colors assigned to each class.

```
% Bulk test
pxdsResults = semanticseg(imdsTest, net, ...
    Classes=classes, MiniBatchSize=4, ...
    WriteLocation=tempdir, Verbose=false);
metrics = evaluateSemanticSegmentation(pxdsResults, pxdsTest);

% Experimental data
I_test = imread(fullfile(testImgDir, testFile));
GT_test = imread(fullfile(testMaskDir, testFile));
% Convert RGB mask → categorical indices if needed
if ndims(GT_test)==3
    GT_label = rgb2label(GT_test, cmap, classes);
else
    GT_label = categorical(GT_test, 1:numel(classes), classes);
end
C_test = semanticseg(I_test, net, Classes=classes);
% Display & IoU
figure;
subplot(1,2,1); imshow(labeloverlay(I_test, C_test, Colormap=cmap));
title('Prediction');
subplot(1,2,2); imshow(labeloverlay(I_test, GT_label, Colormap=cmap));
title('Ground Truth');
iou_test = jaccard(C_test, GT_label);
table(classes(:), iou_test(:), 'VariableNames',{'Class','IoU'})
```

6 Results

The chapter is structured to present, in a systematic manner, the outcomes of training CNN under different turbidity scenarios. As outlined in the project's objectives, the aim is to evaluate the impact of turbidity on the model's ability to perform accurate predictions. Each section corresponds to a specific dataset condition. Chapter 7 provides a critical evaluation of the results obtained, examining the performance of each CNN under different turbidity conditions.

- **No water conditions:** The net is trained under the premise of The network is trained under the premise that the environment does not have water, parameters such as turbidity, water depth, scattering, etc. are not present and therefore the network learns to identify the most relevant characteristics of each object.
- **Low Turbidity:** The second case taken deals with an environment with low turbidity conditions, the properties of water have a slight impact on the visualization of objects.
- **Moderate Turbidity:** To train this network, synthetic data from cases 2 and 3 were combined, as they did not show significant changes. The performance of the neural network is analyzed under moderate to high turbidity conditions.
- **High Turbidity:** Images with a high turbidity content are used; the light absorption and scattering values are high, therefore the visibility is very low, making it difficult to identify objects in the designed synthetic images.
- **Data Fusion:** All the generated data, over 1,800 rendered images and their corresponding segmentation masks, are used to train this network. The network utilizes not only the different degrees of turbidity, but also variations in water color and light.

It is worth mentioning that each synthetic database has 330 images, so this is the number of photos used except for the case of moderate turbidity which has 660 photos. Regarding the results presented, the structure will be identical for each subsection, facilitating network comparisons with the results obtained. The structure is:

- Counting the number of pixels and their distribution graphs.
- Precision and loss graph for synthetic data.
- Randomly display the predictions made for a chosen synthetic image
- Compare the network's predictions to the ground truth images.
- Visualize the intersection over union for the selected synthetic data.

- Evaluate the trained network using the metrics commands.
- Show the predictions made for the images taken in the laboratory during the experiments. Display the metrics for each real-world image. In this chapter, due to the extension of the results, one picture from the experimental phase is shown, the rest are in appendix E.

6.1 No-Water Model

Pixel Count & Distribution

As shown in Table 6.1 and Figure 6.1, the pixel distribution in the No-Water training dataset is heavily imbalanced, with over 80% of all annotated pixels corresponding to the pool's background. In contrast, the remaining object classes particularly the black cylinder are significantly underrepresented, which may introduce class bias during training.

Name	Pixel Count	Image Pixel Count
Pool's Background	147.006.706	180.921.600
Black Cilinder	4.005.385	155.520.000
Silver Cilinder	12.854.311	148.780.800
Wooden Piece	7.968.856	163.296.000
Granite Cobblestone	9.086.342	166.924.800

Table 6.1: Number of pixels in No-Water Model

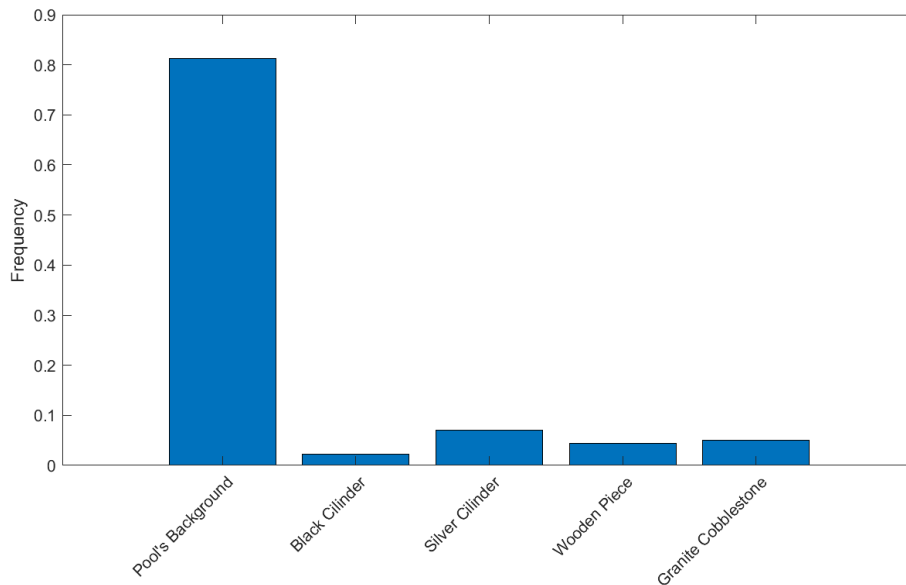


Figure 6.1: Distribution graph for No-Water Model

Synthetic Data: Accuracy & Loss Curves

The No-Water model exhibits rapid convergence, reaching approximately 90% accuracy within the first 200 iterations. Both accuracy and loss curves stabilize early, indicating effective learning in clear conditions.

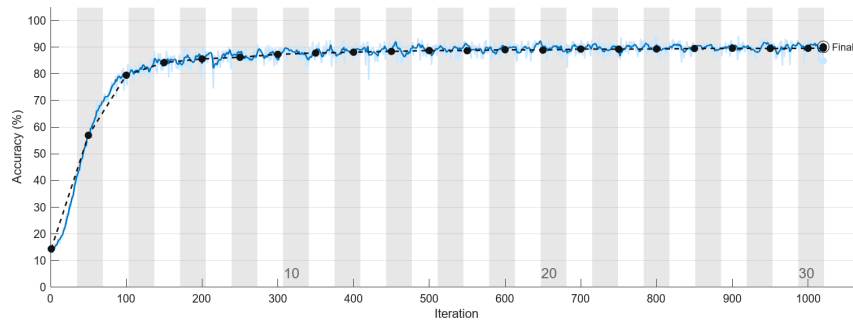


Figure 6.2: Accuracy graph - No Water Network

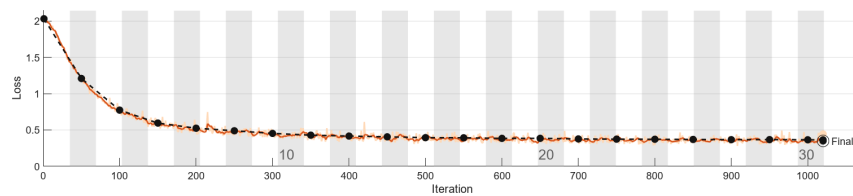


Figure 6.3: Loss curve - No Water Network

Sample Synthetic Prediction

Figure 6.4 is a combination of synthetic photography, truth ground, prediction, and subtraction between prediction and actual values (ground truth). The results of this example image are quite accurate as figure 6.4d reflects. Chapter 7.1.3 explains how to interpret the color range obtained in the fourth image (6.4d) which is a comparison of prediction and ground truth.

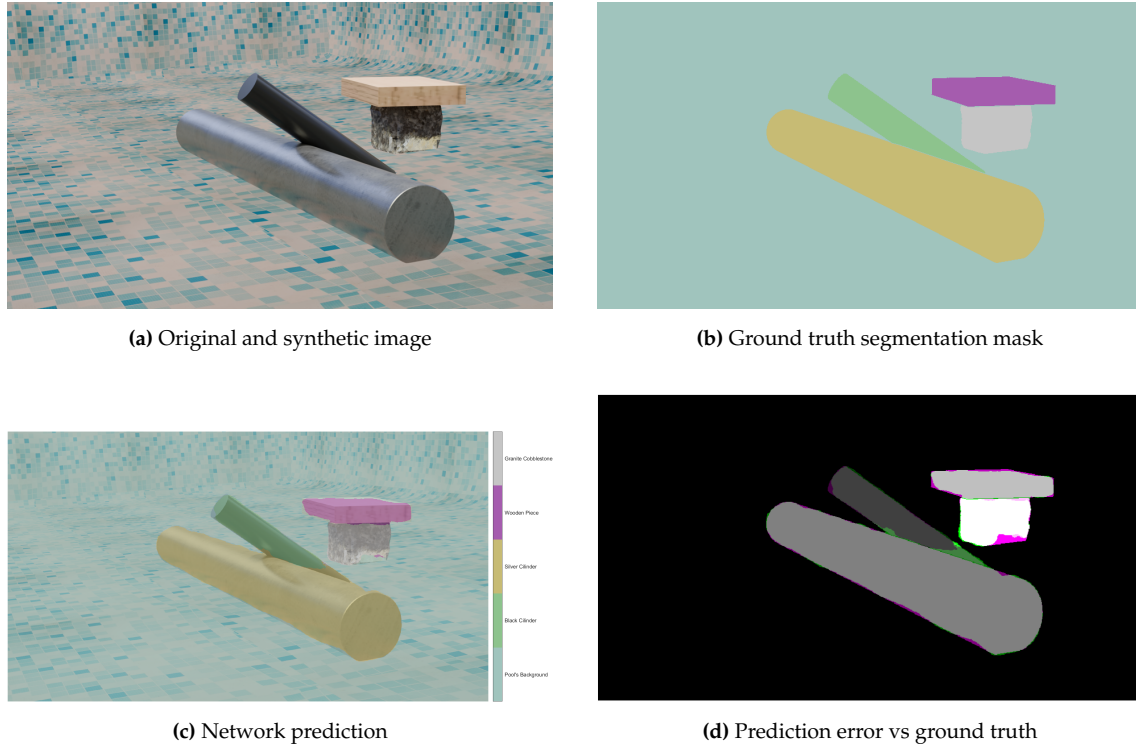


Figure 6.4: Visual comparison of segmentation results for the No-Water Model

Intersection-over-Union Analysis

The IoU results presented in Table 6.2 indicate consistently high performance across all classes, with particularly strong accuracy in segmenting the pool's background and the silver (aluminum) cylinder

Class	IoU
Pool's Background	0.9911
Black Cilinder	0.8699
Silver Cilinder	0.9545
Wooden Piece	0.8971
Granite Cobblestone	0.8766

Table 6.2: IoU per class in random sample No-Water Model

Overall Network Metrics

The evaluation metrics summarized in Tables 6.3 and 6.4 provide a comprehensive overview of the network's segmentation performance. Globally, the model achieves a high overall accuracy of 97.96% and a weighted IoU of 0.9610. Per-class analysis further confirms consistent performance across all categories, with the pool's background and granite cobblestone showing the highest values for IoU and boundary precision.



Figure 6.5: Predictions for Baseline experimental images in No-Water Model

Metrics	GlobalAccuracy	MeanAccuracy	MeanIoU	WeightedIoU	MeanBFScore
Values	0.9796	0.9260	0.8853	0.9610	0.8354

Table 6.3: Global model metrics across the entire dataset in No-Water Model

Clase	Accuracy	IoU	MeanBFScore
Pool's Background	0.9937	0.9824	0.9280
Black Cilinder	0.8547	0.8093	0.7892
Silver Cilinder	0.9312	0.8626	0.7761
Wooden Piece	0.9003	0.8568	0.7722
Granite Cobblestone	0.9503	0.9154	0.8829

Table 6.4: Metrics per class: Accuracy, IoU y Mean Boundary F1 Score

Real-World Image Predictions

The predictions on real-world experimental images are illustrated in Figure 6.5, with Table 6.5 reporting the corresponding Intersection over Union (IoU) scores. Although the network demonstrates moderate success in identifying the pool's background and the black cylinder, the overall IoU average remains low (0.4556), highlighting the challenge of generalizing from synthetic to real data under complex underwater conditions.

Class	IoU
Pool's Background	0.77153
Black Cilinder	0.46452
Silver Cilinder	0.22118
Wooden Piece	0.44961
Granite Cobblestone	0.37868

Table 6.5: IoU values per class for the current experiment, IOU average of classes: 0.4556

6.2 Low-Turbidity Model

Pixel Count & Distribution

In the Low-Turbidity dataset (Table 6.6 and Figure 6.6), the pixel count remains highly imbalanced. The pool background dominates the dataset, representing approximately 80% of the total pixel annotations. Other classes, such as the black cylinder and wooden piece, are underrepresented, which may hinder the network's ability to generalize effectively across all object categories.

Class	PixelCount	ImagePixelCount
PoolBackground	128 113 141	158 630 400
BlackCilinder	3 677 917	138 412 800
SilverCilinder	12 494 623	141 523 200
Wooden Piece	5 744 256	138 412 800
Granite	8 600 463	147 744 000

Table 6.6: Number of pixels in Low-Turbidity Model

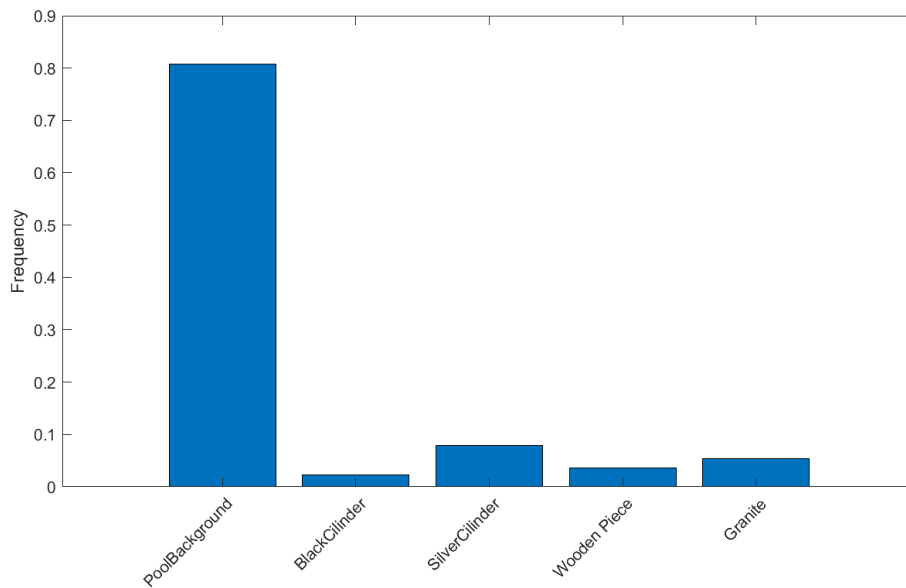


Figure 6.6: Distribution graph for Low-Turbidity Model

Synthetic Data: Accuracy & Loss Curves

The Low-Turbidity model reaches approximately 87% accuracy, with performance stabilizing after 200 iterations. The loss curve shows a gradual decline, indicating consistent training despite the slight presence of turbidity.

6. Results

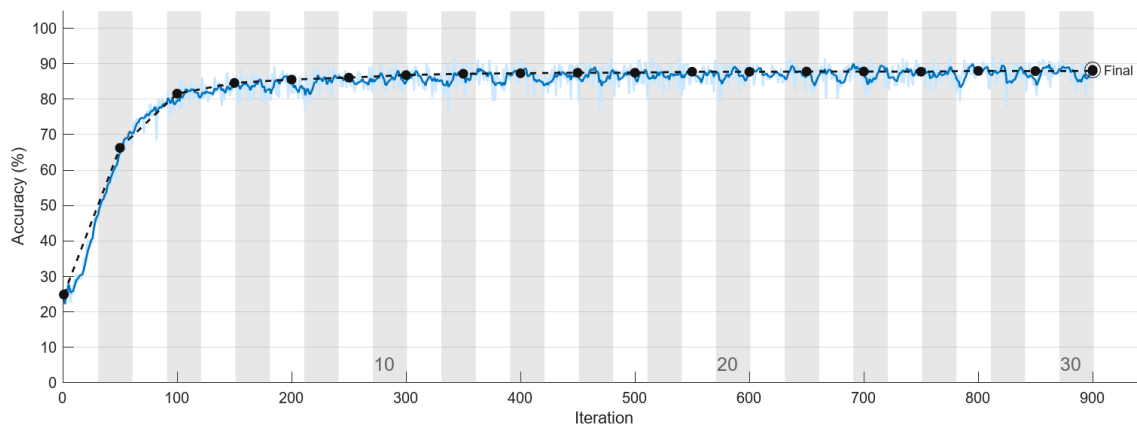


Figure 6.7: Accuracy graph - Low-Turbidity Network

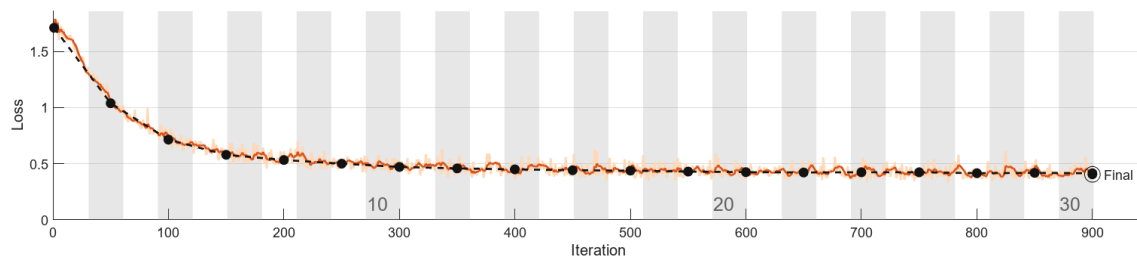


Figure 6.8: Loss curve - Low-Turbidity Network

Sample Synthetic Prediction

The composition 6.24b shows the network performance in the sample image and the difference between the predicted and ground truth, results are slightly inaccurate.

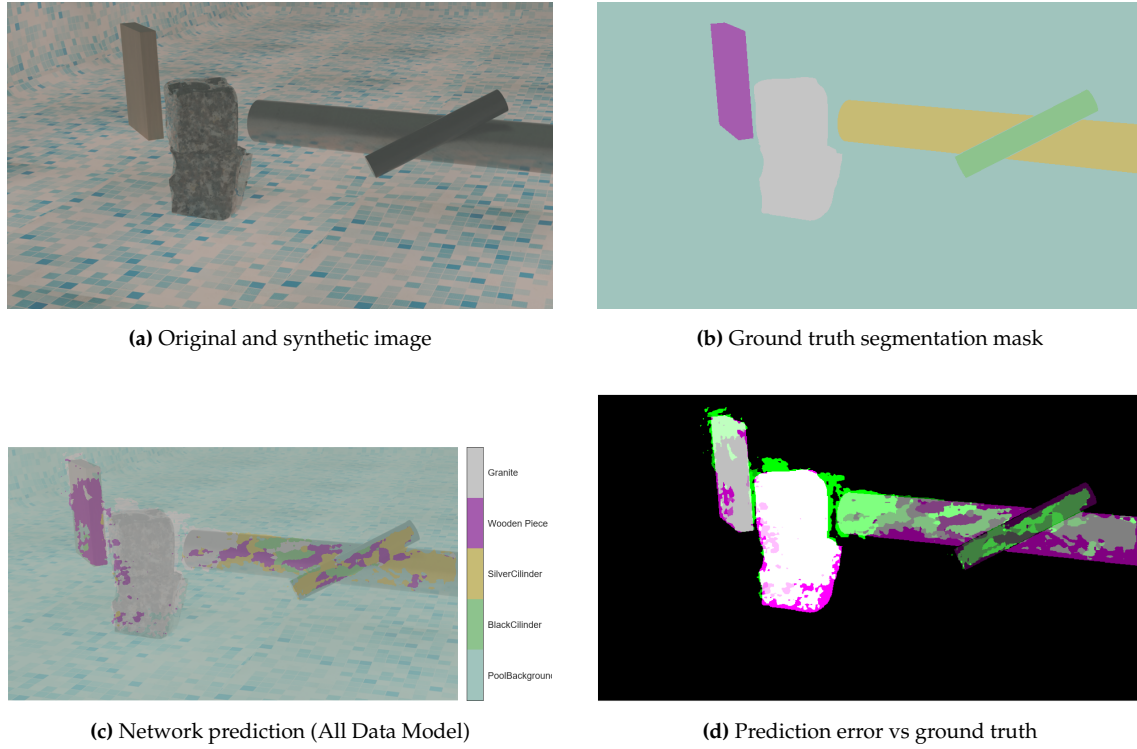


Figure 6.9: Visual comparison of segmentation results for the Low-Turbidity Network.

Intersection-over-Union Analysis

Table 6.7 displays the per-class Intersection over Union (IoU) results for a second real-world experimental test. Notably, the Pool Background class again achieved the highest IoU score (0.9488), indicating consistent performance across experiments. However, other classes such as the Black Cilinder and Silver Cilinder registered much lower scores, highlighting continued challenges in segmenting these objects under real underwater conditions.

Class	IoU
PoolBackground	0.9488
BlackCilinder	0.1101
SilverCilinder	0.2521
Wooden Piece	0.3430
Granite	0.5787

Table 6.7: IoU per class in random sample Low-Turbidity Model

Overall Network Metrics

Table 6.8 presents the global metrics obtained for the entire dataset in this test. Although the Global Accuracy remained high (0.8858), the Mean Accuracy (0.5176) and Mean IoU

(0.4307) reflect considerable imbalance in class-wise performance, as also indicated by the low MeanBFScore (0.3377).

Table 6.9 presents detailed per-class performance metrics for the third real-world experiment. While the PoolBackground class maintained a high accuracy (0.9858) and IoU (0.9138), indicating reliable segmentation, the rest of the classes showed significantly lower performance. The Wooden Piece, in particular, displayed the weakest results across all metrics, with an accuracy of just 0.1460 and a MeanBFScore of 0.1200. These findings confirm that the segmentation network struggles to generalize effectively for certain object classes under real-world conditions.

	GlobalAccuracy	MeanAccuracy	MeanIoU	WeightedIoU	MeanBFScore
Overall	0.8858	0.5176	0.4307	0.8075	0.3377

Table 6.8: Dataset-level metrics for the current experiment in Low-Turbidity Model

Class	Accuracy	IoU	MeanBFScore
PoolBackground	0.9858	0.9138	0.6942
BlackCilinder	0.2365	0.2220	0.2557
SilverCilinder	0.4687	0.3586	0.2428
Wooden Piece	0.1460	0.1232	0.1200
Granite	0.7509	0.5359	0.3078

Table 6.9: Intersection over Union (IoU), Accuracy and Mean Boundary F1 Score per class for the current experiment.

Real-World Image Predictions

Table 6.10 summarizes the per-class IoU scores obtained when evaluating the network trained with low-turbidity synthetic data. The Pool Background class achieves the highest intersection-over-union (0.84364), indicating a reliable prediction. However, all object classes exhibit substantially lower IoU values, with scores around or below 0.25. These results suggest that despite the model performing well on background segmentation, it struggles to accurately detect and segment smaller foreground objects under low turbidity conditions.



Figure 6.10: Predictions for Baseline experimental images using Low-Turbidity Model

Class	IoU
PoolBackground	0.84364
BlackCilinder	0.17000
SilverCilinder	0.17038
Wooden Piece	0.19465
Granite	0.24670

Table 6.10: Per-class Intersection over Union (IoU) for a test image

6.3 Medium-Turbidity Model

Pixel Count & Distribution

As illustrated in Table 6.11 and Figure 6.11, the Medium-Turbidity dataset shows a more balanced pixel distribution compared to previous models. Although the pool background still constitutes the largest proportion of annotations, the other object classes—particularly silver cylinders and granite cobblestones—present significantly higher pixel counts. This improved distribution may offer the network a more diversified training set, potentially enhancing generalization across multiple object categories.

Class	PixelCount	ImagePixelCount
PoolBackground	27 859 384	342 144 000
BlackCilinder	7 668 860	283 046 400
SilverCilinder	25 493 253	291 340 800
Wooden Piece	13 097 387	300 153 600
Granite Cobblestone	17 290 656	317 260 800

Table 6.11: Number of pixels in Medium-Turbidity Model

6. Results

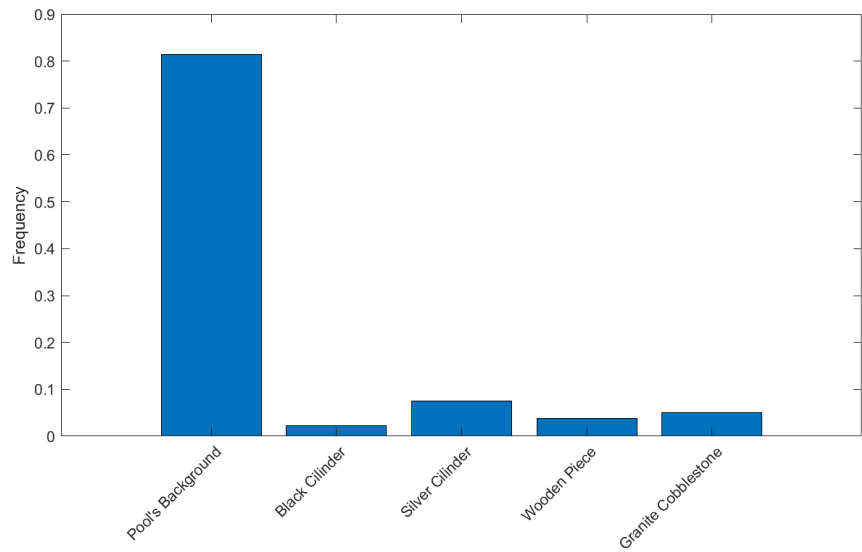


Figure 6.11: Distribution graph for Medium-Turbidity Model

Synthetic Data: Accuracy & Loss Curves

The Medium-Turbidity model achieves an accuracy close to 91%, with convergence reached after approximately 300 iterations. The loss steadily decreases, indicating effective learning despite the increased turbidity in the training dataset.

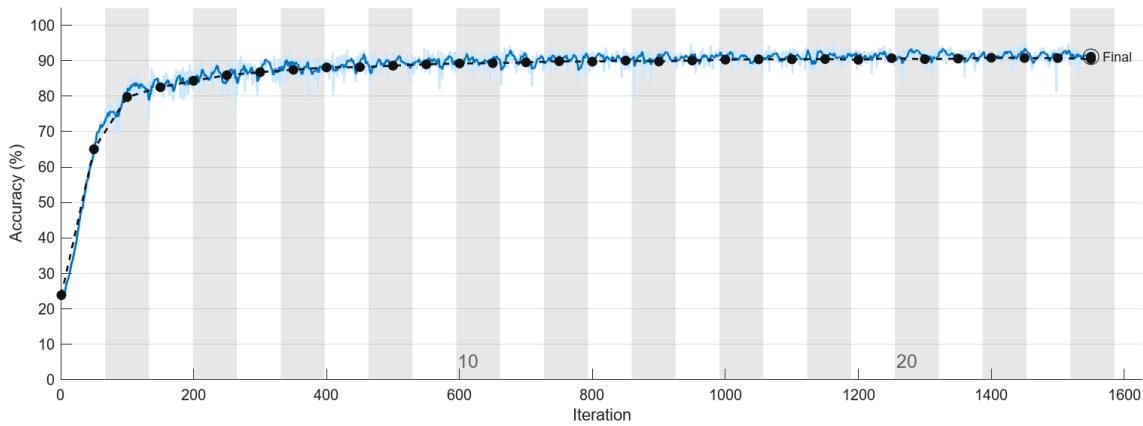


Figure 6.12: Accuracy graph - Medium-Turbidity Model

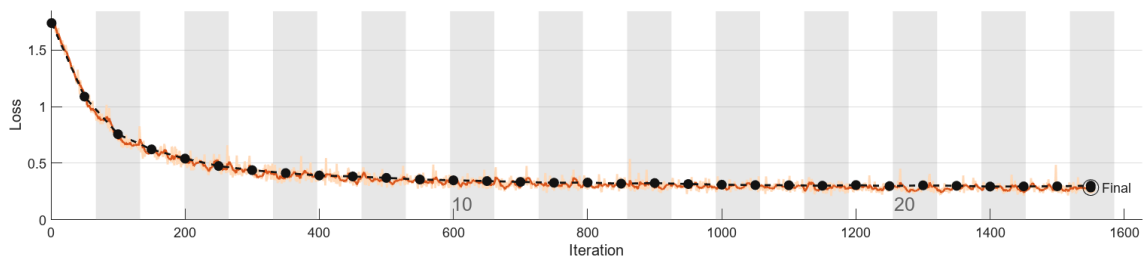


Figure 6.13: Loss curve - Medium-Turbidity Model

Sample Synthetic Prediction

In the medium turbidity scenario, the model shows a tendency to overpredict dominant classes such as Pool Background and Silver Cylinder (aluminum), while significantly underperforming in detecting less represented classes like the Wooden Piece or Granite. The error map highlights frequent false positives and omissions, suggesting that the reduced visibility introduced by turbidity hinders the network's ability to generalize to all object types equally.

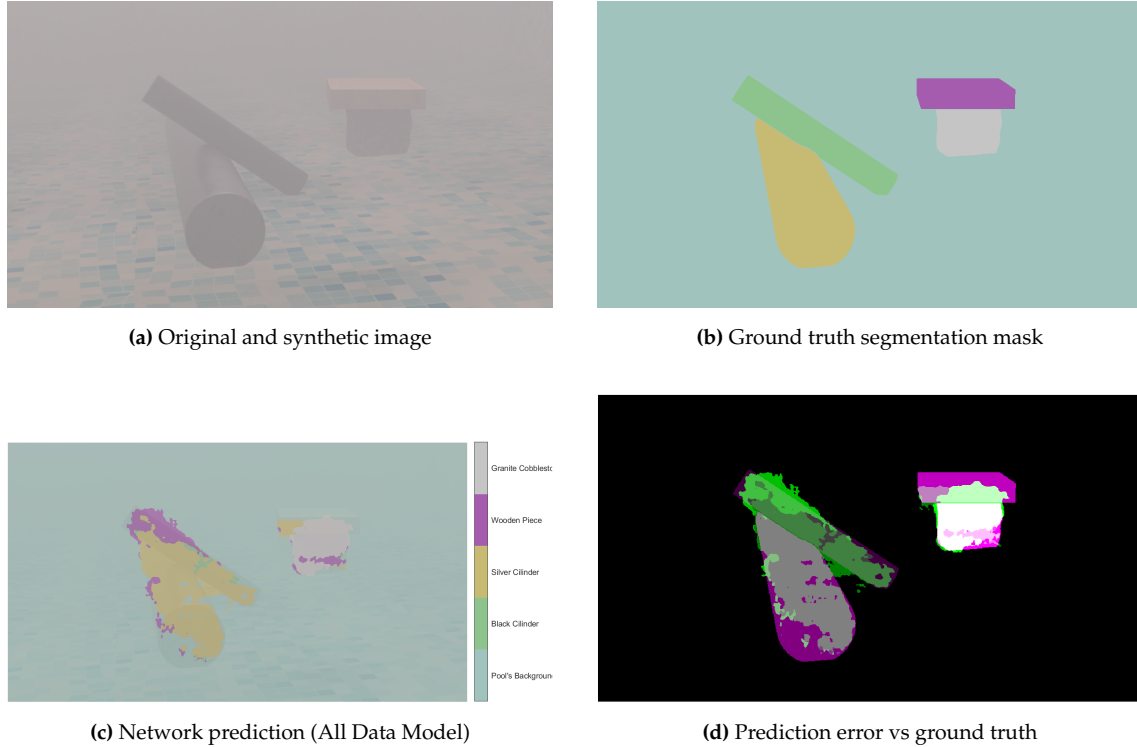


Figure 6.14: Visual comparison of segmentation results for the Medium-Turbidity Model

Intersection-over-Union Analysis

The medium turbidity model performed well in identifying the pool background and granite, but showed poor detection for the wooden piece and black cylinder.

Class	IoU
PoolBackground	0.9614
BlackCylinder	0.0700
SilverCylinder	0.4834
Wooden Piece	0.0017
Granite Cobblestone	0.5415

Table 6.12: Intersection over Union (IoU) per class in the Medium-Turbidity Model

Overall Network Metrics

The medium turbidity network achieved solid global accuracy, with particularly strong performance on the pool background and granite classes, while struggling to consistently segment smaller elements like the wooden and black cylinders.

GlobalAccuracy	MeanAccuracy	MeanIoU	WeightedIoU	MeanBFScore
0.9148	0.6442	0.5641	0.8490	0.4577

Table 6.13: Global performance metrics for the Medium-Turbidity Model

Class	Accuracy	IoU	MeanBFScore
PoolBackground	0.9833	0.9253	0.7372
BlackCilinder	0.3746	0.3026	0.3297
SilverCilinder	0.7144	0.5914	0.3828
Wooden Piece	0.3157	0.2861	0.2348
Granite Cobblestone	0.8328	0.7152	0.4983

Table 6.14: Per-class performance metrics in the Medium-Turbidity Model

Real-World Image Predictions

Under medium turbidity conditions, the network maintained reasonable accuracy on background segmentation, but exhibited notably poor performance across all object classes, particularly for silver and wooden components.

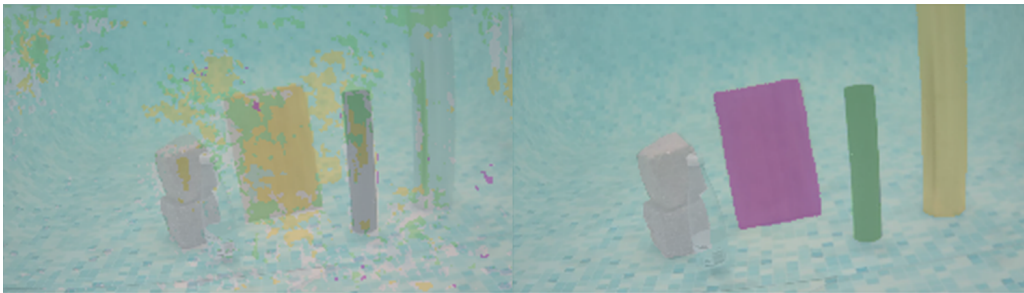


Figure 6.15: Predictions for Baseline experimental images using Medium-Turbidity Model

Class	IoU
PoolBackground	0.7468
BlackCilinder	0.0504
SilverCilinder	0.0062
Wooden Piece	0.0040
Granite	0.2278

Table 6.15: IoU for a test image under Medium-Turbidity

6.4 High-Turbidity Model

In the High-Turbidity dataset, pool background pixels dominate the distribution. The other classes remain significantly underrepresented, mirroring previous patterns of imbalance.

Pixel Count & Distribution

Class	PixelCount	ImagePixelCount
PoolBackground	142 848 171	174 700 800
BlackCilinder	3 866 263	136 857 600
SilverCilinder	12 986 130	147 744 000
Wooden Piece	6 315 440	151 372 800
Granite Cobblestone	8 684 796	161 740 800

Table 6.16: Number of pixels in High-Turbidity Model

6. Results

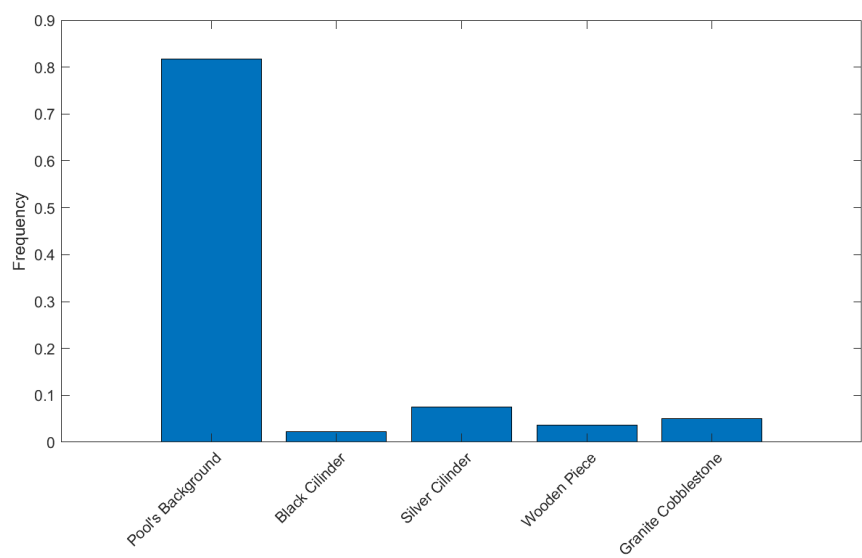


Figure 6.16: Distribution graph for High-Turbidity Model

Synthetic Data: Accuracy & Loss Curves

The High-Turbidity model reaches an accuracy of approximately 86% with slower convergence compared to previous cases. The loss curve remains relatively high, suggesting challenges in learning under severe visibility degradation.

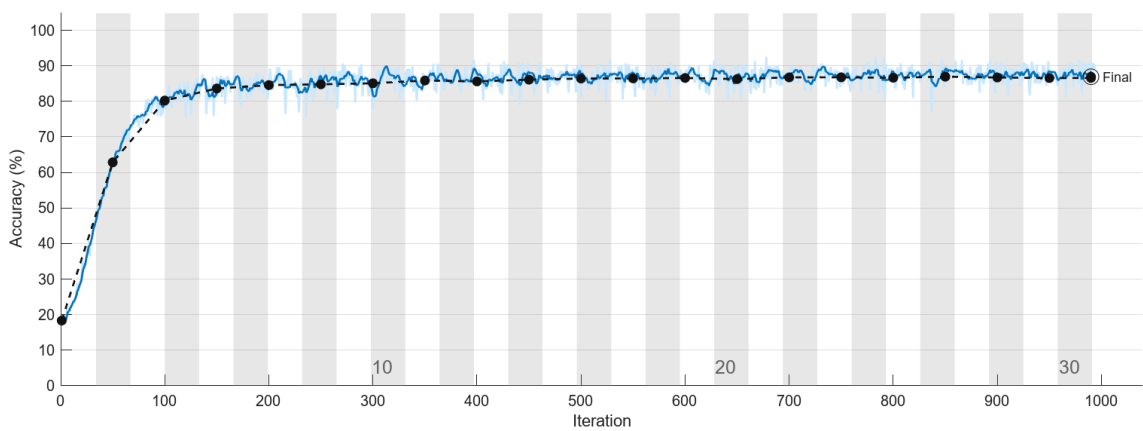


Figure 6.17: Accuracy graph - High-Turbidity Model

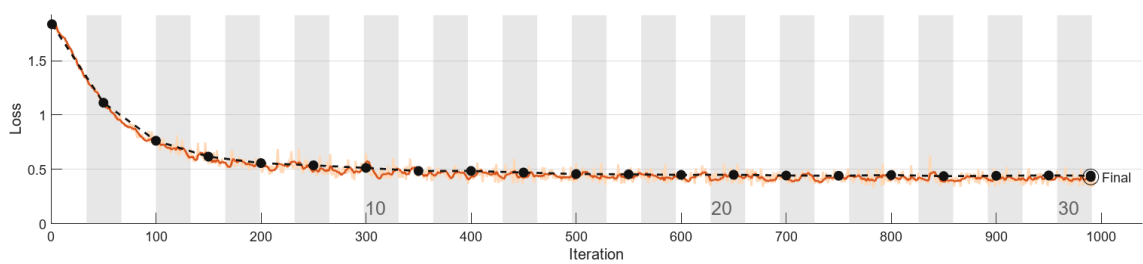


Figure 6.18: Loss curve - High-Turbidity Model

Sample Synthetic Prediction

In high turbidity conditions, segmentation accuracy significantly declined, particularly for non-background classes, with notable confusion between similarly shaped objects and frequent overprediction artifacts.

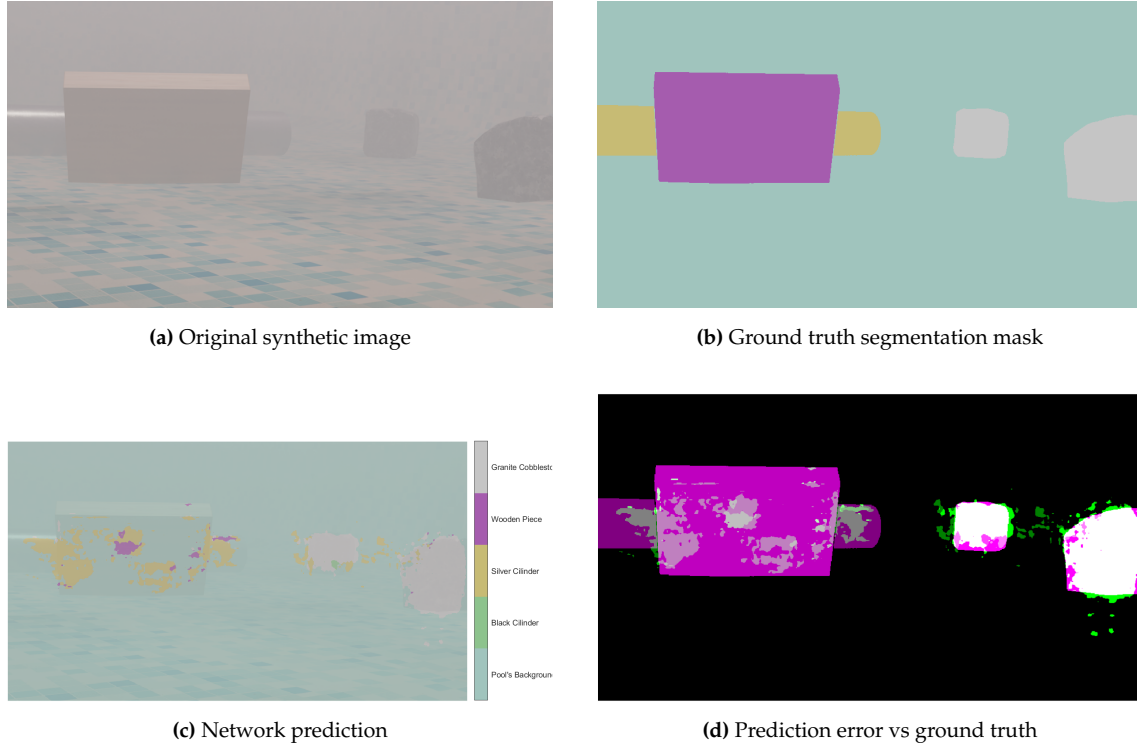


Figure 6.19: Visual comparison of segmentation results for the High-Turbidity Model

Intersection-over-Union Analysis

The absence of the black cylinder in the image justifies the IoU value of 0.0000. The background and granite cobblestone classes achieved high scores, while the remaining objects were either poorly segmented or largely omitted.

Class	IoU
PoolBackground	0.8650
BlackCilinder	0.0000
SilverCilinder	0.1384
Wooden Piece	0.0285
Granite Cobblestone	0.7706

Table 6.17: IoU for each class in the High-Turbidity Model

Overall Network Metrics

The segmentation model trained on the high-turbidity dataset exhibits strong performance for the background and granite classes, while the remaining objects—especially the black cylinder and wooden piece—show significantly lower accuracy and boundary scores, indicating difficulty in feature extraction under poor visibility.

GlobalAccuracy	MeanAccuracy	MeanIoU	WeightedIoU	MeanBFScore
0.8757	0.4475	0.3773	0.7865	0.3026

Table 6.18: Global performance metrics of the segmentation model

Class	Accuracy	IoU	MeanBFScore
PoolBackground	0.9848	0.8943	0.6417
BlackCilinder	0.1027	0.0920	0.1206
SilverCilinder	0.3346	0.2469	0.1831
Wooden Piece	0.0605	0.0574	0.1164
Granite Cobblestone	0.7547	0.5961	0.3369

Table 6.19: Per-class performance metrics of the segmentation model

Real-World Image Predictions

Under high turbidity conditions, the network struggled to accurately identify most objects, with the exception of the background class, which still retained a moderate IoU score.



Figure 6.20: Predictions for Baseline experimental images using High-Turbidity Model

6.5 Full Dataset Model

Pixel Count & Distribution

The All-Images network exhibits a strong class imbalance, with the pool background dominating the dataset, which may bias the training process toward this majority class.

Class	PixelCount	ImagePixelCount
PoolBackground	76 419 542	949 190 400
BlackCilinder	20 341 832	809 222 400
SilverCilinder	74 675 106	832 550 400
Wooden Piece	41 894 708	853 286 400
Granite Cobblestone	48 083 332	884 908 800

Table 6.20: Number of pixels in the Full Dataset Model

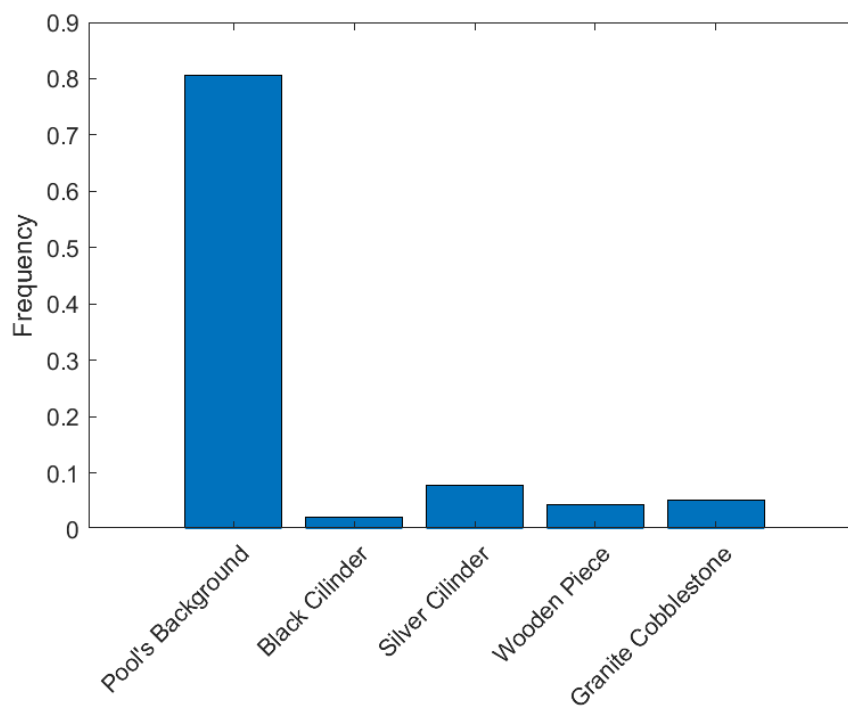


Figure 6.21: Distribution graph for Full Dataset Model

Synthetic Data: Accuracy & Loss Curves

The All-Images Network achieved the highest accuracy, reaching approximately 92%, with a smooth and stable learning curve. The loss steadily decreased, indicating consistent convergence despite the greater data variability.

6. Results

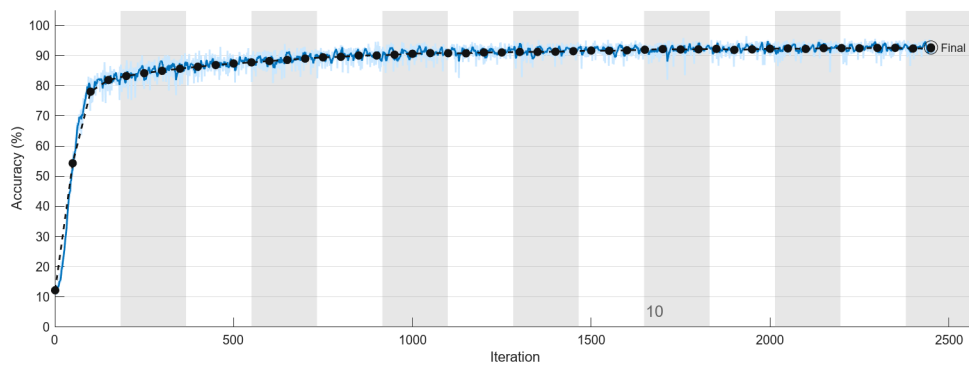


Figure 6.22: Accuracy graph - Full Dataset Model

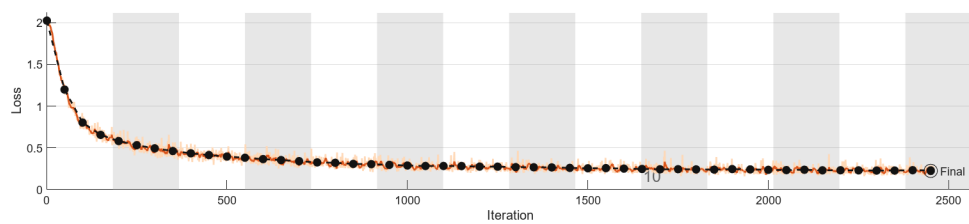


Figure 6.23: Loss curve - Full Dataset Model

Sample Synthetic Prediction

The prediction for the Full-Dataset Model shows reasonable segmentation performance, although misclassification around object boundaries is evident, especially between the silver and black cylinders.

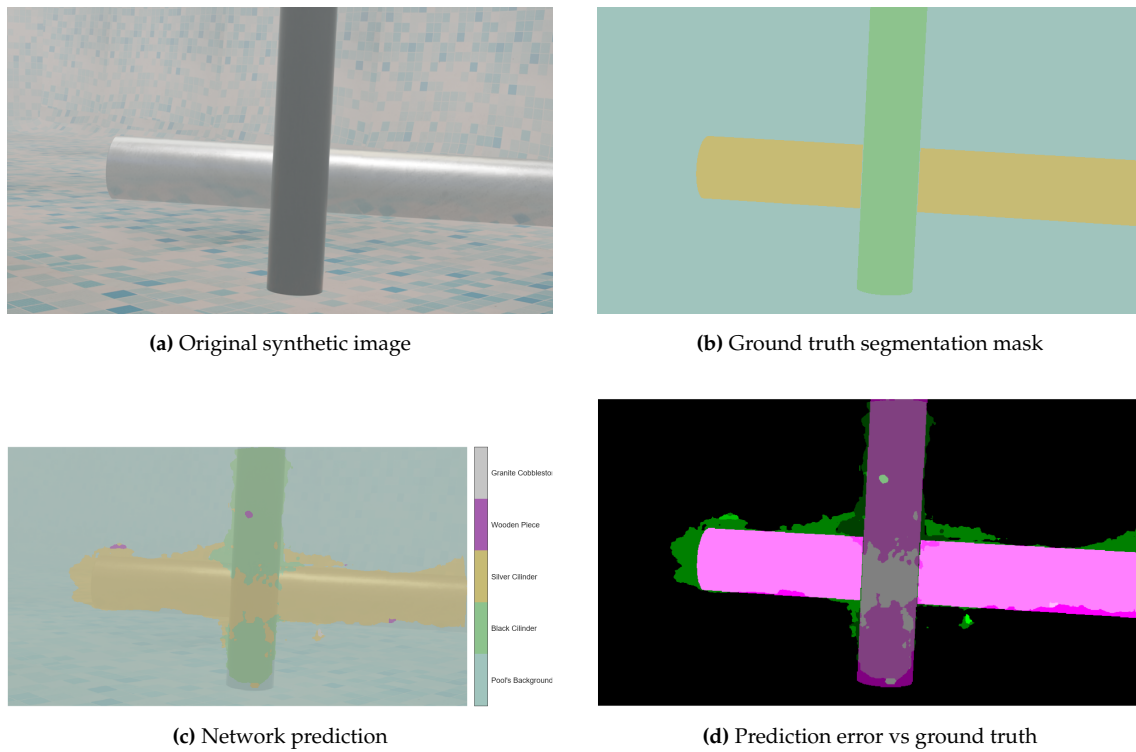


Figure 6.24: Visual comparison of segmentation results for the Full Dataset Model

Intersection-over-Union Analysis

The Full Dataset Model shows good segmentation, identifies the classes presented in the picture.

Class	IoU
PoolBackground	0.9363
BlackCilinder	0.6739
SilverCilinder	0.7182
Wooden Piece	0.0000
Granite Cobblestone	0.0000

Table 6.21: Intersection over Union (IoU) for each class in the Full Dataset Model

Overall Network Metrics

The Full Dataset Model demonstrates the most balanced performance across all classes, achieving both high global accuracy and strong per-class IoU scores, particularly for PoolBackground, Granite, and SilverCylinder.

GlobalAccuracy	MeanAccuracy	MeanIoU	WeightedIoU	MeanBFScore
0.9241	0.7150	0.6259	0.8670	0.4975

Table 6.22: Global performance metrics of the Full Dataset Model

Class	Accuracy	IoU	MeanBFScore
PoolBackground	0.9811	0.9372	0.7644
BlackCilinder	0.4208	0.3465	0.3713
SilverCilinder	0.7399	0.5986	0.4213
Wooden Piece	0.6113	0.5182	0.3184
Granite Cobblestone	0.8220	0.7292	0.5307

Table 6.23: Per-class performance metrics of the Full Dataset Model

Real-World Image Predictions

The All-Data Network performs consistently well across all objects in the selected real-world test image, with the background and wooden piece achieving the highest IoU values.

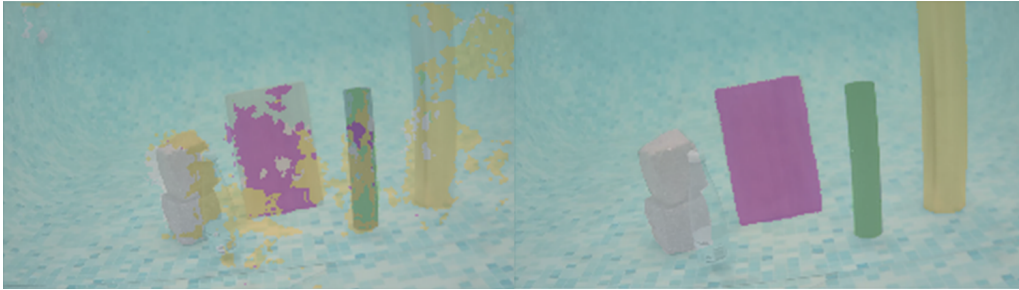


Figure 6.25: Predictions for Baseline experimental images using Full Dataset Model

Class	IoU
PoolBackground	0.87337
BlackCilinder	0.31697
SilverCilinder	0.23102
Wooden Piece	0.50075
Granite Cobblestone	0.38605

Table 6.24: Per-class Intersection over Union (IoU) on a single test image

7 Discussion

This chapter presents an analysis of the results obtained and discusses the influence of varying turbidity levels on the training performance of the neural networks.

7.1 Comparison between neural networks

7.1.1 Pixel distribution

Regarding the pixel distribution across classes (figures 6.1 , 6.6 , 6.11 , 6.16 and 6.21), it is evident that the dataset is imbalanced. The pool background appears by far the most frequently, followed by the aluminum pipe. Although this may initially seem insignificant, it becomes highly relevant when analyzing prediction errors. As will be discussed later, the networks tend to over predict the most frequent classes in the dataset, particularly these two, while failing to correctly identify others. In contrast, classes with fewer visible pixels—such as the granite block and the black cylinder are significantly less likely to be recognized in the real-world test images.

7.1.2 Accuracy and loss curves

The accuracy and loss graphs obtained during training, were generated exclusively from synthetic data. These results are summarized in Table 7.1. Figure 6.2, 6.7, 6.12 , 6.17 and Figure 6.22 illustrate the training accuracy curves for each dataset. Two main trends can be observed:

On one hand, there is a pattern between the number of training images and model performance—datasets with more samples lead to higher accuracy and lower loss. For example, although the Medium Turbidity and High Turbidity models are trained under more challenging visual conditions due to increased scattering, the model trained on Medium Turbidity images achieves 91% accuracy, outperforming Low Turbidity, which reaches only 87%. This might be attributed to the Medium Turbidity dataset containing nearly twice as many training images as the Low Turbidity dataset.

On the other hand, training under turbid conditions does not necessarily result in improved performance on similarly turbid images. In fact, as turbidity increases, model accuracy tends to decline, as seen in the High Turbidity network, which achieved the lowest accuracy (86%) among all models.

Finally, the All Data model—trained on the complete dataset spanning all turbidity levels achieved the best results overall, with the highest accuracy (92%) and lowest loss (0.24).

This indicates that a diversified training set helps the network generalize better across varying underwater conditions.

Model Type	Accuracy [%]	Loss
No Water	90	0.4
Low Turbidity	87	0.45
Medium Turbidity	91	0.38
High Turbidity	86	0.5
All Data	92	0.24

Table 7.1: Accuracy and loss values obtained from training with each dataset

7.1.3 Comparison Between Predicted and Ground Truth Masks

To qualitatively evaluate the model performance, a composite image was generated by overlapping the predicted segmentation mask with the ground truth mask. This results are illustrated in 6.4 , 6.9 , 6.14 , 6.19 and 6.24. Each pixel in the image is assigned a color based on the agreement or disagreement between the predicted and true label, allowing for intuitive visual interpretation of the model's errors and successes.

The color scheme follows a false-color convention, where specific hues and intensities represent different semantic relationships between the two masks. These include correct matches, false positives, and false negatives. In addition, color intensity encodes the relative class value, with lighter tones indicating higher classes. Table ?? summarizes the visual meaning of each color and its interpretation:

The color-coded output obtained when comparing predicted segmentation masks with ground truth annotations visually highlights the agreement or disagreement between them. Below is the explanation of each color and its meaning:

- **Black:** Represents background pixels correctly classified as background in both the predicted and ground truth masks.
- **Dark to Light Gray:** Indicates pixels that belong to the same object class in both masks. Lighter shades represent higher class indices.
- **Green:** Marks pixels that are labeled as belonging to an object class in the ground truth but were not identified by the model. This is a false negative.
- **Green shades (Green scale):** The brighter the green, the higher the class index that was missed. This helps to assess which specific classes are more frequently overlooked.
- **Magenta:** Highlights pixels that the model predicted as an object class, but which are not actually present in the ground truth. This is a false positive.

- **Magenta shades (Magenta scale):** Brighter shades of magenta indicate overpredictions of higher class indices. This reveals which false classes are being incorrectly predicted.
- **White:** Indicates a perfect match for higher-index classes, meaning the object was correctly identified at a detailed class level.

This visualization serves as an essential diagnostic tool for understanding which objects or regions are more prone to misclassification, and whether the model tends to underpredict or overpredict certain classes. It also highlights class imbalance effects and segmentation difficulties, especially in cases involving visually similar or low-contrast objects.

7.1.4 Intersection-over-Union Analysis

This metric was used selectively to assess the prediction quality on specific test images. Intersection-over-Union (IoU) quantifies the overlap between the predicted segmentation and the ground truth for a given class. It is defined as the ratio between the number of correctly predicted pixels for a particular class (intersection) and the total number of pixels belonging to that class in either the prediction or the ground truth (union). Therefore, the higher the IoU value, the more accurate the prediction for that class in the analyzed image. The IoU is not a valid tool to represent the overall performance of a network if it is not used for the entire set of images.

7.1.5 Metrics

Figures 6.3, 6.8, 6.13, 6.18 and 6.22 support the findings discussed in subsection 7.1.2. Global accuracy is higher when the dataset contains lower turbidity levels. Even though the number of training samples also has a significant impact on performance. The data suggest that increasing the dataset size may not enhance accuracy as effectively as training under clear water conditions.

7.1.6 Prediction on the real images

Overall, when evaluating all the predicted images, the results are considerably poor. The best predictions were achieved by the model trained on the dataset without turbidity. As turbidity increases, a clear decline in prediction accuracy can be observed.

There are some studies that have investigated object detection under foggy conditions. The YOLOv8x model was evaluated on the basis of its performance in various fog densities by Faiz et al. [60]. They found that its performance declined as fog density increased, which aligns with the project finding that the lower the turbidity, the higher the detection accuracy. Additionally, they noted that the results were more reliable as the dataset's diversity increased. This supports the observation expressed in section 7.1.2 that a larger number of training samples improves accuracy. Zhang and Jia [61] proposed HR-YOLO,

an improved detection model designed specifically for foggy environments. According to their findings, the accuracy of detection can be enhanced by enhancing the quality of the image. The absence of turbidity has a more significant impact on the accuracy than the number of training images, which is consistent with the analysis of the carried experiments during this project. Lastly, Chu et al. [60] introduced D-YOLO, a dual-path detection network that integrates both hazy and dehazed features through an attention-based fusion module. This discovery reflects the ideas expressed in this study, as models developed on clarity-free datasets notably outperformed those built on unclear data, regardless of sample quantity.

7.2 Impact of turbidity on training

The model trained with the complete dataset appears to suffer from overfitting. Regarding the impact of turbidity, as previously discussed in Section 7.1.3, there is evidence to suggest that turbidity negatively affects the performance of the network.

In terms of the training process, notable differences were observed in the training times required for each network. Specifically, the models trained on the moderate turbidity dataset and the full dataset required significantly more time to complete training. This may be attributed to differences in dataset composition or size.

For the other models, a trend was observed in which higher turbidity levels made the training process more challenging, suggesting that increasing turbidity complicates feature extraction and convergence during learning.

8 Conclusion

Regarding the objective of the network’s performance evaluation, as outlined in the motivation of the project (see Section 1.3), this goal has been successfully achieved. A convolutional neural network (CNN) was trained under various predefined turbidity levels, and its performance was critically assessed in Chapter 7 through a structured analysis.

A clear trend was observed: training the network under clear water conditions—i.e., in the absence of turbidity—enables better feature extraction, leading to more accurate object identification. This advantage remains even when the model is later exposed to scenes with reduced visibility, where object features are partially obscured by suspended particles.

When tested on synthetic data, all trained models demonstrated strong performance, achieving high accuracy in identifying objects within simulated underwater scenes. However, a significant drop in accuracy was noted when these models were applied to real-world images with varying turbidity levels. The underlying reasons for this degradation are further discussed in Section 7.

Despite this limitation, synthetic imagery proves to be a viable and scalable alternative for training segmentation models in scenarios where annotated real-world data is scarce. With proper refinement, this approach can effectively support underwater perception tasks in visually challenging environments.

A primary constraint of the study was the limited availability of time and computational resources, which restricted the exploration of alternative network architectures that could potentially enhance model performance.

9 FutureWork

This chapter outlines several recommendations and directions that could be explored to further develop and improve the project.

If the work is continued, one of the first steps would be to experiment with different training configurations. This could involve adjusting parameters such as the maximum number of epochs, the minibatch size, or particularly modifying the class weight balancing strategies. Exploring these adjustments may lead to better convergence and more balanced prediction results across classes.

A more in-depth analysis of class-wise performance is also advisable. Tools such as Matlab's `allScores` function could be employed to gain insights into the contribution and influence of each class on the final segmentation output. This type of analysis may help identify dominant classes in each image and relate them to those used during training. Furthermore, deliberate manual tuning of class weights could be implemented to mitigate the dominance of overrepresented classes—such as the pool background and the aluminum pipe, which were the most frequent in the dataset. Rebalancing the weight distribution might help the model better differentiate less-represented classes, such as the black cylinder or granite blocks, and reduce bias toward background classification.

Another promising pathway would be to explore alternative neural network architectures. Beginning with object detection models such as YOLO and its variants referenced in Chapter 8 could provide a complementary approach to semantic segmentation, potentially improving accuracy under varying visual conditions.

Regarding the synthetic data pipeline, improvements could be made by enhancing the simulation of underwater conditions. Investigating more advanced material and light interaction models would contribute to generating more realistic underwater imagery. Additionally, performing new absorbance measurements using quartz cuvettes (which are more suitable for low-wavelength light) could reveal whether optical properties of water samples significantly influence network training or image realism.

Lastly, if a shift in the methodology is considered, alternative sensing techniques such as sonar could be tested. Sonar systems are less affected by water turbidity and might provide more robust object detection in challenging underwater environments. Similarly, the use of hyperspectral cameras could be explored to determine whether they can extract additional spectral features that enhance object classification capabilities in low-visibility conditions.

Bibliography

- [1] T. T. Bengtsson, M. Frederiksen, and J. E. Larsen, "The Danish welfare state: A sociological investigation", *The Danish Welfare State: A Sociological Investigation*, pp. 1–269, Jan. 2016.
- [2] E. A. Gøsta, *The Three Worlds of Welfare Capitalism*, 1990. [Online]. Available: https://www.saxo.com/dk/the-three-worlds-of-welfare-capitalism-gosta-esping-andersen-paperback_9780745607962.
- [3] Danish Maritime Authority, *Bekendtgørelse om sikkerhed på offshoreanlæg (Offshore Safety Regulation), Consolidated Act No. 125/2018*, 2018. [Online]. Available: <https://www.retsinformation.dk/eli/lta/2018/125>.
- [4] International Labour Organization, "Merchant Shipping (Minimum Standards) Convention, 1976 (ILO C147)", ILO – International Labour Organization, Tech. Rep., 1976.
- [5] Y. Y. Schechner and N. Karpel, "Clear Underwater Vision", in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2004.
- [6] G. K. I. M. K. Y. A. L. Chen L. C.; Papandreou, "DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs", *arXiv preprint arXiv:1606.00915*, 2017. [Online]. Available: <https://arxiv.org/abs/1606.00915>.
- [7] *URPC2020: Underwater Robotic Vision Dataset*. [Online]. Available: <https://www.kaggle.com/datasets/irvingvasquez/underwater-robotic-vision-urpc2020>.
- [8] M. J. Islam, Y. Xia, and J. Sattar, "Semantic Segmentation of Underwater Imagery: Dataset and Benchmark", in *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.
- [9] M. J. Islam, Y. Luo, and J. Sattar, *Simultaneous Enhancement and Super-resolution of Underwater Imagery for Improved Visual Perception*. 2020.
- [10] M. A. Shafique *et al.*, "Deep Learning-based Semantic Segmentation of Underwater Scenes Using Synthetic Data", vol. 22, 2022.
- [11] C. Mai, C. Wiele, J. Liniger, and S. Pedersen, "Synthetic subsea imagery for inspection under natural lighting with marine-growth", *Ocean Engineering*, vol. 313, p. 119284, Dec. 2024. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0029801824026222#b9>.
- [12] C. Mai, J. Liniger, and S. Pedersen, "Semantic segmentation using synthetic images of underwater marine-growth", *Frontiers in Robotics and AI*, vol. 11, p. 1459570, Jan. 2024.
- [13] P. N. Stuart Russell, *Artificial Intelligence: A Modern Approach*, Third edition. Prentice Hall, 2010.

- [14] Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning: An Introduction*, Second. MIT Press, 2014.
- [15] I Goodfellow, Y Bengio, A Courville, and Y Bengio, *Deep learning*. MIT Press, 2016. [Online]. Available: <https://synapse.koreamed.org/pdf/10.4258/hir.2016.22.4.351>.
- [16] F. Rosenblatt, "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain", *Psychological Review*, vol. 65, pp. 386–408, 1958.
- [17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *Advances in Neural Information Processing Systems (NeurIPS)*. Curran Associates, Inc., 2012, vol. 25, pp. 1097–1105.
- [18] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition", *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2016-December, pp. 770–778, Dec. 2015. [Online]. Available: <https://arxiv.org/pdf/1512.03385>.
- [19] Yann LeCun and Léon Bottou and Yoshua Bengio and Patrick Haffner, "Gradient-based learning applied to document recognition", *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, 1998.
- [20] Karen Simonyan and Andrew Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition", *arXiv*, 2014.
- [21] J. Y. S. P. R. S. Z. D. Z. Szegedy C. Liu W., "Going Deeper with Convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.
- [22] Abheer Bhandodker, *ResNet Architecture Explained. Prior to the advent of ReNet*. [Online]. Available: <https://medium.com/@abheerchrome/resnet-architecture-explained-53347e169185>.
- [23] *Deep Learning Toolbox Model for ResNet-18 Network - File Exchange - MATLAB Central*. [Online]. Available: <https://es.mathworks.com/matlabcentral/fileexchange/68261-deep-learning-toolbox-model-for-resnet-18-network>.
- [24] *tubos fluorescentes t5*. [Online]. Available: <https://es.rs-online.com/web/c/iluminacion/bombillas-lamparas-y-tubos/tubos-fluorescentes/?if=tubos-fluorescentes-t5>.
- [25] *Bestway Power Steel pool | 404 x 201 x 100 cm | with filter pump and accessories | Pool.shop*. [Online]. Available: https://www.pool.shop/p/bestway-power-steel-pool-404-x-201-x-100-cm-including-pump-and-pool-stairs/?utm_source=chatgpt.com.
- [26] *Kaolin USP testing specifications meets 1332-58-7*. [Online]. Available: <https://www.sigmaaldrich.com/DK/en/product/sial/k1512>.
- [27] *OceanTools : C3 Compact Subsea SD, HD and IP Video Camera*. [Online]. Available: <https://www.oceantools.co.uk/cameras/c3-compact-subsea-camera>.

- [28] *Neutrally Buoyant ROV Tether for BlueROV2 and Other Subsea Vehicles*. [Online]. Available: <https://bluerobotics.com/store/cables-connectors/cables/fathom-rov-tether-rov-ready/>.
- [29] *Pinout of PoE (Power over Ethernet) Cables | Basler Product Documentation*. [Online]. Available: <https://docs.baslerweb.com/knowledge/pinout-of-poe-power-over-ethernet-cables>.
- [30] *Ethernet Cables - RJ45/Colors & Crossover*. [Online]. Available: https://buy.advantech.eu/CMS/CmsDetail.aspx?CMSID=B50A4401-6634-498A-82FA-076F63093B97&CMSType=White_Papers.
- [31] *Ubiquiti U-POE-AF PoE Injector 2x RJ45 Strøminjektor | Billig*. [Online]. Available: <https://www.proshop.dk/PoE-injektor/Ubiquiti-U-POE-AF-PoE-Injector-2x-RJ45/2789105>.
- [32] *TL-SG105PE | Easy Smart Switch Gigabit de 5 puertos con PoE + de 4 puertos | TP-Link España*. [Online]. Available: <https://www.tp-link.com/es/business-networking/poe-switch/tl-sg105pe/>.
- [33] *SoftPerfect Network Scanner : fast, flexible, advanced*. [Online]. Available: <https://www.softperfect.com/products/networkscanner/>.
- [34] *SADP - HiTools - Hikvision Europe*. [Online]. Available: <https://www.hikvision.com/europe/support/tools/hitools/clea8b3e4ea7da90a9/>.
- [35] *OceanTools Ltd : World Leading Underwater Technologists, Subsea IRM Engineering, Off-shore ROV Survey*. [Online]. Available: <https://www.oceantools.co.uk/>.
- [36] *Official download of VLC media player, the best Open Source player - VideoLAN*. [Online]. Available: <https://www.videolan.org/vlc/>.
- [37] *MATLAB - El lenguaje del cálculo técnico*. [Online]. Available: <https://es.mathworks.com/products/matlab.html>.
- [38] Curtis D. Mobley, *Light and Water: Radiative Transfer in Natural Waters*. Academic Press, 1944.
- [39] John T. O. Kirk, *Light and Photosynthesis in Aquatic Ecosystems*. Cambridge University Press, 1994.
- [40] *Versatile, Routine UV-Vis Instrument, Cary 60 | Agilent*. [Online]. Available: https://www.agilent.com/en/product/molecular-spectroscopy/uv-vis-uv-vis-nir-spectroscopy/uv-vis-uv-vis-nir-systems/cary-60-uv-vis-spectrophotometer?utm_source=chatgpt.com.
- [41] G. M. F. H. C. A. B. G. O. N. H. Marcel Babin David Stramski, "Variations in the light absorption coefficients of phytoplankton, nonalgal particles, and dissolved organic matter in coastal waters around Europe", *Journal of Geophysical Research: Oceans*, vol. 108 (C7), 2003.
- [42] *Extraction Kits*. [Online]. Available: https://www.agilent.com/store/en_US/Prod-5610-2039/5610-2039.

- [43] *Visible Light - NASA Science*. [Online]. Available: https://science.nasa.gov/ems/09_visiblelight/.
- [44] *Plastic Disposable Cuvettes (Cells), Polystyrene, UV-Vis Cuvette | Agilent*. [Online]. Available: <https://www.agilent.com/en/product/molecular-spectroscopy/uv-vis-uv-vis-nir-spectroscopy/uv-vis-uv-vis-nir-cuvettes-flow-cells/disposable-polystyrene-cells-for-uv-vis-uv-vis-nir-systems>.
- [45] *UV Cuvettes, also called Quartz Cuvettes*. [Online]. Available: <https://www.advaluetech.com/products/lab-supplies/uv-quartz-cuvettes/>.
- [46] *Photopea | Online Photo Editor*. [Online]. Available: <https://www.photopea.com/>.
- [47] *blender.org - Home of the Blender project - Free and Open 3D Creation Software*. [Online]. Available: <https://www.blender.org/>.
- [48] R. Atienza, "Improving Model Generalization by Agreement of Learned Representations from Data Augmentation", *Proceedings - 2022 IEEE/CVF Winter Conference on Applications of Computer Vision, WACV 2022*, pp. 3927–3936, Oct. 2021. [Online]. Available: <https://arxiv.org/pdf/2110.10536>.
- [49] C. Shorten and T. M. Khoshgoftaar, "A survey on Image Data Augmentation for Deep Learning", *Journal of Big Data*, vol. 6, no. 1, pp. 1–48, Dec. 2019. [Online]. Available: <https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0197-0>.
- [50] *Rock And Stone Brushes For Blender - Superhive (formerly Blender Market)*. [Online]. Available: <https://superhivemarket.com/products/rock-and-stone-brushes-for-blender>.
- [51] *Family Oasis 3D Rectangular Pool 3D Model - TurboSquid 2079634*. [Online]. Available: <https://www.turbosquid.com/3d-models/family-oasis-3d-rectangular-pool-2079634>.
- [52] *BlenderKit | FREE 3D models, textures and other Blender assets*. [Online]. Available: <https://www.blenderkit.com/>.
- [53] *Poliigon - PBR Textures, Models and HDRIs*. [Online]. Available: <https://www.poliigon.com/>.
- [54] *Polished Granite Stone Texture, Grey - Poliigon*. [Online]. Available: <https://www.poliigon.com/texture/polished-granite-stone-texture-grey/1117>.
- [55] *Light Objects - Blender 4.4 Manual*. [Online]. Available: https://docs.blender.org/manual/en/latest/render/lights/light_object.html.
- [56] *Tutorial: Cycles Volume Scatter Explained*. [Online]. Available: <https://blenderdiplom.com/en/tutorials/all-tutorials/582-cycles-volume-scatter.html>.
- [57] *Semantic Segmentation Using Deep Learning - MATLAB & Simulink*. [Online]. Available: <https://es.mathworks.com/help/vision/ug/semantic-segmentation-using-deep-learning.html>.

- [58] *trainingOptions* - Opciones para entrenar una red neuronal de deep learning - MATLAB. [Online]. Available: <https://es.mathworks.com/help/deeplearning/ref/trainingoptions.html>.
- [59] *deeplabv3plus* - Create DeepLab v3+ convolutional neural network for semantic image segmentation - MATLAB. [Online]. Available: <https://es.mathworks.com/help/vision/ref/deeplabv3plus.html>.
- [60] T. M. G. Faiz Muhammad; Ahmad, "Object Detection in Foggy Weather using Deep Learning Model", *ResearchGate*, 2025. [Online]. Available: https://www.researchgate.net/publication/388753437_Object_Detection_in_Foggy_Weather_using_Deep_Learning_Model.
- [61] Yifan Zhang and Ning Jia, "A target detection model HR-YOLO for advanced driver assistance systems in foggy conditions", *Scientific Reports*, vol. 15, 2025. [Online]. Available: <https://www.nature.com/articles/s41598-025-98286-4>.

A Appendix - ResNet-18 Architecture

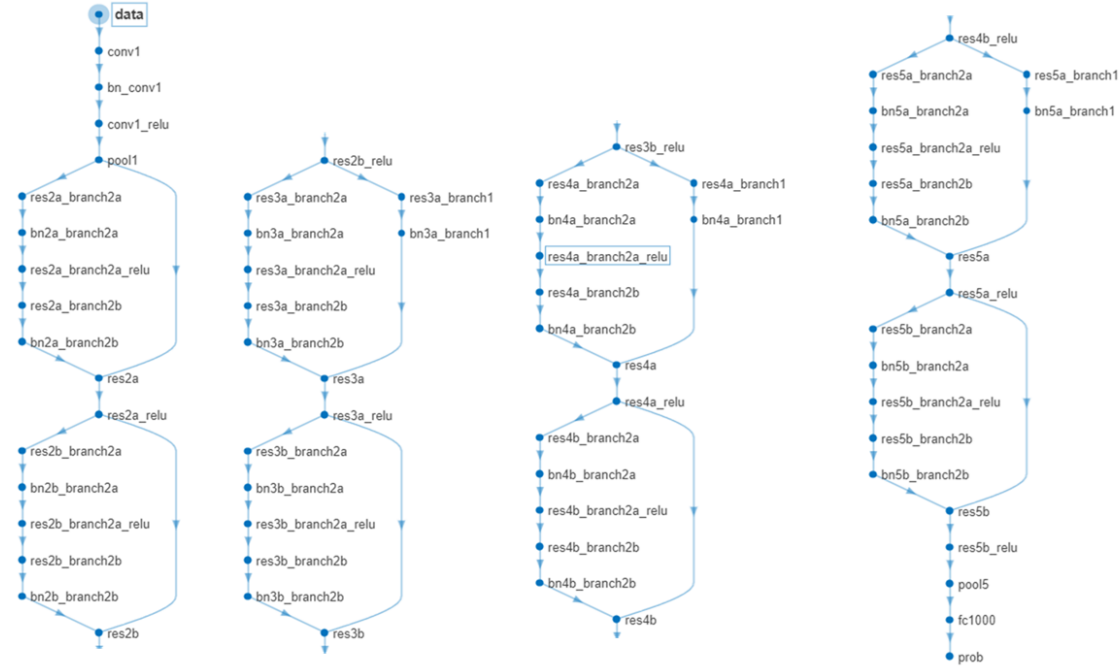


Figure A.1: ResNet-18 Diagram

Table A.1: ResNet-18 Architecture: Layers 1–20

No.	Layer name	Type	Activations	Learnable parameters
1	data	Image Input	224×224×3×1	—
2	conv1	2-D Convolution	112×112×64×1	weights: 7×7×3×64; bias: 1×1×64
3	bn_conv1	Batch Norm	112×112×64×1	scale, offset: 1×1×64 each
4	conv1_relu	ReLU	112×112×64×1	—
5	pool1	2-D Max Pooling	56×56×64×1	—
6	res2a_branch2a	2-D Convolution	56×56×64×1	weights: 3×3×64×64; bias: 1×1×64
7	bn2a_branch2a	Batch Norm	56×56×64×1	scale, offset: 1×1×64 each
8	res2a_branch2a_relu	ReLU	56×56×64×1	—
9	res2a_branch2b	2-D Convolution	56×56×64×1	weights: 3×3×64×64; bias: 1×1×64
10	bn2a_branch2b	Batch Norm	56×56×64×1	scale, offset: 1×1×64 each
11	res2a	Addition	56×56×64×1	—
12	res2a_relu	ReLU	56×56×64×1	—

(Continued on next page)

A. Appendix - ResNet-18 Architecture

(Continued from previous page)

No.	Layer name	Type	Activations	Learnable parameters
13	res2b_branch2a	2-D Convolution	56×56×64×1	weights: 3×3×64×64; bias: 1×1×64
14	bn2b_branch2a	Batch Norm	56×56×64×1	scale, offset: 1×1×64 each
15	res2b_branch2a_relu	ReLU	56×56×64×1	—
16	res2b_branch2b	2-D Convolution	56×56×64×1	weights: 3×3×64×64; bias: 1×1×64
17	bn2b_branch2b	Batch Norm	56×56×64×1	scale, offset: 1×1×64 each
18	res2b	Addition	56×56×64×1	—
19	res2b_relu	ReLU	56×56×64×1	—
20	res3a_branch2a	2-D Convolution	28×28×128×1	weights: 3×3×64×128; bias: 1×1×128
21	bn3a_branch2a	Batch Norm	28×28×128×1	scale, offset: 1×1×128 each
22	res3a_branch2a_relu	ReLU	28×28×128×1	—
23	res3a_branch2b	2-D Convolution	28×28×128×1	weights: 3×3×128×128; bias: 1×1×128
24	bn3a_branch2b	Batch Norm	28×28×128×1	scale, offset: 1×1×128 each
25	res3a_branch1	2-D Convolution	28×28×128×1	weights: 1×1×64×128; bias: 1×1×128
26	bn3a_branch1	Batch Norm	28×28×128×1	scale, offset: 1×1×128 each
27	res3a	Addition	28×28×128×1	—
28	res3a_relu	ReLU	28×28×128×1	—
29	res3b_branch2a	2-D Convolution	28×28×128×1	weights: 3×3×128×128; bias: 1×1×128
30	bn3b_branch2a	Batch Norm	28×28×128×1	scale, offset: 1×1×128 each
31	res3b_branch2a_relu	ReLU	28×28×128×1	—
32	res3b_branch2b	2-D Convolution	28×28×128×1	weights: 3×3×128×128; bias: 1×1×128
33	bn3b_branch2b	Batch Norm	28×28×128×1	scale, offset: 1×1×128 each
34	res3b	Addition	28×28×128×1	—
35	res3b_relu	ReLU	28×28×128×1	—
36	res4a_branch2a	2-D Convolution	14×14×256×1	weights: 3×3×128×256; bias: 1×1×256
37	bn4a_branch2a	Batch Norm	14×14×256×1	scale, offset: 1×1×256 each
38	res4a_branch2a_relu	ReLU	14×14×256×1	—
39	res4a_branch2b	2-D Convolution	14×14×256×1	weights: 3×3×256×256; bias: 1×1×256
40	bn4a_branch2b	Batch Norm	14×14×256×1	scale, offset: 1×1×256 each
41	res4a_branch1	2-D Convolution	14×14×256×1	weights: 1×1×128×256; bias: 1×1×256
42	bn4a_branch1	Batch Norm	14×14×256×1	scale, offset: 1×1×256 each
43	res4a	Addition	14×14×256×1	—
44	res4a_relu	ReLU	14×14×256×1	—

(Continued on next page)

A. Appendix - ResNet-18 Architecture

(Continued from previous page)

No.	Layer name	Type	Activations	Learnable parameters
45	res4b_branch2a	2-D Convolution	14×14×256×1	weights: 3×3×256×256; bias: 1×1×256
46	bn4b_branch2a	Batch Norm	14×14×256×1	scale, offset: 1×1×256 each
47	res4b_branch2a_relu	ReLU	14×14×256×1	—
48	res4b_branch2b	2-D Convolution	14×14×256×1	weights: 3×3×256×256; bias: 1×1×256
49	bn4b_branch2b	Batch Norm	14×14×256×1	scale, offset: 1×1×256 each
50	res4b	Addition	14×14×256×1	—
51	res4b_relu	ReLU	14×14×256×1	—
52	res5a_branch2a	2-D Convolution	7×7×512×1	weights: 3×3×256×512; bias: 1×1×512
53	bn5a_branch2a	Batch Norm	7×7×512×1	scale, offset: 1×1×512 each
54	res5a_branch2a_relu	ReLU	7×7×512×1	—
55	res5a_branch2b	2-D Convolution	7×7×512×1	weights: 3×3×512×512; bias: 1×1×512
56	bn5a_branch2b	Batch Norm	7×7×512×1	scale, offset: 1×1×512 each
57	res5a_branch1	2-D Convolution	7×7×512×1	weights: 1×1×256×512; bias: 1×1×512
58	bn5a_branch1	Batch Norm	7×7×512×1	scale, offset: 1×1×512 each
59	res5a	Addition	7×7×512×1	—
60	res5a_relu	ReLU	7×7×512×1	—
61	res5b_branch2a	2-D Convolution	7×7×512×1	weights: 3×3×512×512; bias: 1×1×512
62	bn5b_branch2a	Batch Norm	7×7×512×1	scale, offset: 1×1×512 each
63	res5b_branch2a_relu	ReLU	7×7×512×1	—
64	res5b_branch2b	2-D Convolution	7×7×512×1	weights: 3×3×512×512; bias: 1×1×512
65	bn5b_branch2b	Batch Norm	7×7×512×1	scale, offset: 1×1×512 each
66	res5b	Addition	7×7×512×1	—
67	res5b_relu	ReLU	7×7×512×1	—
68	pool5	Global Avg Pooling	1×1×512×1	—
69	fc1000	Fully Connected	1×1×1000×1	weights: 1000×512; bias: 1000×1
70	prob	Softmax	1×1×1000×1	—

B Appendix - Segmentation Mask Generation

```
import bpy
import os
import math

def render_segmentacion_y_raw(output_dir, paisaje_objs, mats_paisaje,
matsegmentacion, cam_positions):
    # Create the paths if don't exist
    mascara_dir = os.path.join(output_dir, "mascara_segmentacion")
    raw_dir = os.path.join(output_dir, "raw_imagenes")
    os.makedirs(mascara_dir, exist_ok=True)
    os.makedirs(raw_dir, exist_ok=True)

    scene = bpy.context.scene
    cam = scene.camera

    # Get the next frame number automatically
    def get_next_index(folder):
        existing = [f for f in os.listdir(folder) if f.endswith(".png")]
        if not existing:
            return 1
        existing_nums = [int(f.split(".")[0]) for f in existing if
        f.split(".")[0].isdigit()]
        return max(existing_nums) + 1 if existing_nums else 1

    idx = get_next_index(mascara_dir)

    for pos, rot in cam_positions:
        frame_name = f"{idx:04d}"

        Assign position and rotation to the camera
        cam.location = pos
        cam.rotation_euler = rot

    #1 creates the Segmentation render
    for obj in paisaje_objs:
        obj.active_material = matsegmentacion
        scene.render.filepath = os.path.join(mascara_dir, frame_name + ".png")
        bpy.ops.render.render(write_still=True)
```


B. Appendix - Segmentation Mask Generation

```
#2 creates the original/raw images
for obj, mat in zip(paisaje_objs, mats_paisaje):
    obj.active_material = mat
    scene.render.filepath = os.path.join(raw_dir, frame_name + ".png")
    bpy.ops.render.render(write_still=True)

    idx += 1 # Aumenta el índice del nombre del archivo

# Sets the output folder
output_dir = r"C:\Users\mikio\Desktop\Pool"

# Sets the objects that will appear
paisaje_objs = [
    bpy.data.objects["BlackCilinder"],
    bpy.data.objects["SilverCilinder"],
    bpy.data.objects["WoodenPiece"],
    bpy.data.objects["GraniteCobblestone1"],
    bpy.data.objects["GraniteCobblestone2"],
    bpy.data.objects["GraniteCobblestone3"],
    bpy.data.objects["PoolsBackground"],
    bpy.data.objects["Water"]
]

# Sets the material to the objects
mats_paisaje = [
    bpy.data.materials["mat11"],
    bpy.data.materials["Frozen white metal"],
    bpy.data.materials["Natural Pine Wood"],
    bpy.data.materials["mat4"],
    bpy.data.materials["mat4"],
    bpy.data.materials["mat4"],
    bpy.data.materials["PBackground"],
    bpy.data.materials["WaterProp"]
]

# Sets the segmentation mask to the objects
matsegmentacion = bpy.data.materials["matsegmentacion"]

Define the positions XYZ and XYZ rotation in radians
cam_positions = [
    ([2.1876, -0.139, -0.13949],
    [math.radians(85.15), math.radians(-2.686), math.radians(-633.03)]),
    ([2.1876, -0.56223, -0.13949],
    [math.radians(85.15), math.radians(-2.686), math.radians(-642.61)]),
```

```
([2.1876, -1.0051, -0.13949],  
[math.radians(85.15), math.radians(-2.686), math.radians(-653.14)]),  
([1.2814, -1.0051, -0.13949],  
[math.radians(85.15), math.radians(-2.686), math.radians(-665.83)]),  
([0.24339, -1.0051, -0.13949],  
[math.radians(85.15), math.radians(-2.686), math.radians(-715.38)]),  
([-0.71151, -1.0051, -0.13949],  
[math.radians(85.15), math.radians(-2.686), math.radians(-766.87)]),  
([-2.1785, -1.0051, -0.13949],  
[math.radians(85.15), math.radians(-2.686), math.radians(-789.95)]),  
([-2.1785, -0.62995, -0.13949],  
[math.radians(85.712), math.radians(-2.7883), math.radians(-799.67)]),  
([-2.1785, -0.12293, -0.13949],  
[math.radians(85.712), math.radians(-2.7883), math.radians(-809.73)]),  
([-2.1785, 0.64807, -0.13949],  
[math.radians(85.712), math.radians(-2.7883), math.radians(-826.21)]),  
([-1.3023, 0.64807, -0.13949],  
[math.radians(85.712), math.radians(-2.7883), math.radians(-831.83)]),  
([-0.45946, 0.64807, -0.13949],  
[math.radians(85.712), math.radians(-2.7883), math.radians(-857.93)]),  
([0.78394, 0.64807, -0.13949],  
[math.radians(85.712), math.radians(-2.7883), math.radians(-951.53)]),  
([1.7773, 0.64807, -0.13949],  
[math.radians(85.712), math.radians(-2.7883), math.radians(-968.4)]),  
([1.7773, 0.64807, 0.24793],  
[math.radians(73.534), math.radians(2.0628), math.radians(-968.32)]),  
([1.7773, 0.19052, 0.24793],  
[math.radians(73.534), math.radians(2.0628), math.radians(-980.51)]),  
([1.7773, -0.3208, 0.020469],  
[math.radians(79.728), math.radians(3.0444), math.radians(-998.87)]),  
([1.6333, -0.77644, 0.020469],  
[math.radians(79.728), math.radians(3.0444), math.radians(-1011.9)]),  
([1.0748, -0.77644, 0.29043],  
[math.radians(66.506), math.radians(-8.1847), math.radians(-1029.9)]),  
([0.017811, -0.77644, 0.29043],  
[math.radians(55.576), math.radians(-10.962), math.radians(-1093.5)]),  
([-1.5262, -0.22958, 0.55828],  
[math.radians(64.856), math.radians(0.8171), math.radians(-1165.5)]),  
([-1.5262, 0.20207, 0.55828],  
[math.radians(64.856), math.radians(0.8171), math.radians(-1176.3)]),  
([-1.5262, -0.77644, 0.55828],  
[math.radians(63.924), math.radians(-25.467), math.radians(-1138.8)]),  
([-1.5262, 0.89005, 0.31416],
```

B. Appendix - Segmentation Mask Generation

```
[math.radians(74.138), math.radians(6.0765), math.radians(-1199.2)],
([-0.82002, 0.89005, 0.31416],
[math.radians(66.398), math.radians(-2.1693), math.radians(-1216.9)],
([-0.033676, 0.89005, 0.079797],
[math.radians(71.494), math.radians(-3.572), math.radians(-1244.7)],
([0.98131, 0.89005, 0.079797],
[math.radians(71.494), math.radians(-3.572), math.radians(-1297.3)],
([1.7491, 0.61135, 0.079797],
[math.radians(79.007), math.radians(-7.6286), math.radians(-1321.9)],
([1.079, 0.20459, 0.079797],
[math.radians(79.007), math.radians(-7.6286), math.radians(-1336.8)],
([0.21656, -0.56372, 0.079797],
[math.radians(57.65), math.radians(-6.3956), math.radians(-1435.4)],
([-0.50165, -0.56372, 0.079797],
[math.radians(70.281), math.radians(-14.318), math.radians(-1498.7)],
([-0.059662, 0.48936, 0.032715],
[math.radians(70.281), math.radians(-14.318), math.radians(-1225.5)],
([0.2708, 0.48936, 0.58282],
[math.radians(161.41), math.radians(-190.18), math.radians(-1228.4)])
]
```

Object Placement I have deleted most of them, the idea is the same:
to put positions following the same structure.

```
composiciones_objetos = [
    {
        "SilverCilinder": {"loc": (-0.3918, 0.47566, 0.13896),
        "rot": (math.radians(0), math.radians(0), math.radians(0))},
        "BlackCilinder": {"loc": (0.57164, -0.31983, -0.1512),
        "rot": (math.radians(0), math.radians(0), math.radians(0))},
        "GraniteCobblestone3": {"loc": (0.56957, -0.34484, 0.060706),
        "rot": (math.radians(0), math.radians(0), math.radians(-54.679))},
        "GraniteCobblestone1": {"loc": (0.023351, -0.076065, -0.25806),
        "rot": (math.radians(87.992), math.radians(1.4775), math.radians(-387.02))},
        "WoodenPiece": {"loc": (0.12894, 0.39159, -0.2343),
        "rot": (math.radians(45), math.radians(0), math.radians(-64.699))},
    },
    {
        "SilverCilinder": {"loc": (0.20467, 0.43373, -0.26387),
        "rot": (math.radians(90), math.radians(0), math.radians(69.808))},
        "BlackCilinder": {"loc": (0.57164, -0.33989, -0.026217),
        "rot": (math.radians(177.33), math.radians(0), math.radians(0))},
        "GraniteCobblestone3": {"loc": (0.56957, -0.32478, -0.23906),
        "rot": (math.radians(178.46), math.radians(2.1788), math.radians(54.65))},
    }
]
```

```
"GraniteCobblestone1": {"loc": (0.85093, -0.000336, 0.25806),
"rot": (math.radians(87.992), math.radians(1.4775), math.radians(-387.02))},
"WoodenPiece": {"loc": (0.16886, 0.37313, -0.2343),
"rot": (math.radians(45), math.radians(0), math.radians(159.81))},
},
{
"SilverCilinder": {"loc": (0.24358, 0.090395, -0.22191),
"rot": (math.radians(265.37), math.radians(180.57), math.radians(173))},
"BlackCilinder": {"loc": (0.57164, 0.045368, -0.16616),
"rot": (math.radians(0), math.radians(0), math.radians(0))},
"GraniteCobblestone3": {"loc": (-0.10497, 0.46522, -0.23906),
"rot": (math.radians(178.46), math.radians(2.1788), math.radians(54.65))},
"GraniteCobblestone1": {"loc": (0.12824, -0.003301, -0.21342),
"rot": (math.radians(0), math.radians(90), math.radians(90))},
"WoodenPiece": {"loc": (0.16886, -0.002515, -0.29092),
"rot": (math.radians(90), math.radians(0), math.radians(10))},
},
]
1

# Apply a composition to objects in the scene
def aplicar_composicion(compo_dict):
    for nombre_objeto, valores in compo_dict.items():
        obj = bpy.data.objects[nombre_objeto]
        obj.location = valores["loc"]
        obj.rotation_euler = valores["rot"]

# executes the function in the loop
def main():
    for composicion in composiciones_objetos:
        aplicar_composicion(composicion)
        render_segmentacion_y_raw(output_dir, paisaje_objs,
        mats_paisaje, matsegmentacion, cam_positions)
main()
```

C Appendix - Training Procedure Using ResNet-18

```
rutaImagenes = 'C:\Users\mikio\Desktop\EntrenamientoT1\CamVid\images';  
rutaEtiquetas = 'C:\Users\mikio\Desktop\EntrenamientoT1\CamVid\labels';
```

```
classes = [  
    "Pool_background"  
    "Black_Pipe"  
    "Silver_Pipe"  
    "Wooden_Box"  
    "Granite"  
];
```

```
labelIDs = [  
    160 196 189; % Pool's background/Fondo de piscina  
    141 195 141; % Black cilinder/ Cilindro pequeño negro  
    199 187 116; % Silver Cilinder/ Cilindro grande plateado  
    165 92 174; % Wooden box/ Madero  
    197 197 197 % Granite/ Granito  
];
```

```
imds = imageDatastore(rutaImagenes);  
pxds = pixelLabelDatastore(rutaEtiquetas, classes, labelIDs);
```

```
numFiles = numel(imds.Files);
```

```
randIdx = randperm(numFiles);
```

```
trainRatio = 0.8;  
numTrain = round(trainRatio * numFiles);
```

```
trainIdx = randIdx(1:numTrain);  
valIdx = randIdx(numTrain+1:end);
```

```
imdsTrain = subset(imds, trainIdx);
imdsVal    = subset(imds, valIdx);

pxdsTrain = subset(pxds, trainIdx);
pxdsVal    = subset(pxds, valIdx);

trainingData = combine(imdsTrain, pxdsTrain);

validationData = combine(imdsVal, pxdsVal);

imageSize = [540, 960, 3]; % height, width and channels
numClasses = 5;
network = 'resnet18';

options = trainingOptions('sgdm', ...
    'InitialLearnRate',1e-4, ...
    'MaxEpochs',30, ...
    'MiniBatchSize',6, ...
    'Shuffle','every-epoch', ...
    'VerboseFrequency',10, ...
    'Plots','training-progress', ...
    'ExecutionEnvironment','auto');

lgraph = deeplabv3plusLayers([540 960 3],5,'resnet18');
[net, info] = trainNetwork(trainingData, lgraph, options);

class(net)

save("miRedEntrenadaT1num2.mat","net","info");
```

D Appendix - Semantic Segmentation Using DeepLab v3+

```
classes = getClassNames()

imagePretrainedNetwork("resnet18")

%Main folder where both folders are
outputFolder = fullfile("C:\Users\mikio\Desktop\EntrenamientoT1", "CamVid");

% Define routes directly
imgDir = fullfile(outputFolder, "images");
labelDir = fullfile(outputFolder, "labels");

imgDir = fullfile(outputFolder, "images");
imds = imageDatastore(imgDir);

I = readimage(imds,25);
I = histeq(I)

figure
imshow(I)
impixelinfo

labelIDs = camvidPixelLabelIDs();

labelDir = fullfile(outputFolder,"labels");
pxds = pixelLabelDatastore(labelDir,classes,labelIDs);

C = readimage(pxds,25);
cmap = camvidColorMap;
B = labeloverlay(I,C,ColorMap=cmap);
imshow(B)
pixelLabelColorbar(cmap,classes);

tbl = countEachLabel(pxds) %count the number of pixels by class label

frequency = tbl.PixelCount/sum(tbl.PixelCount);
```

```
bar(1:numel(classes),frequency)
xticks(1:numel(classes))
xticklabels(tbl.Name)
xtickangle(45)
ylabel("Frequency")

%now Prepare Training, Validation, and Test Sets

[imdsTrain, imdsVal, imdsTest, pxdsTrain, pxdsVal, pxdsTest] =
partitionCamVidData(imds,pxds);

numTrainingImages = numel(imdsTrain.Files)

numValImages = numel(imdsVal.Files)

numTestingImages = numel(imdsTest.Files)

dsVal = combine(imdsVal,pxdsVal);

dsTrain = combine(imdsTrain,pxdsTrain);

xTrans = [-10 10];
yTrans = [-10 10];
dsTrain = transform(dsTrain, @(data)augmentImageAndLabel(data,xTrans,yTrans));

%Create the Network

imageSize = [960 540 3]; %mi data set size with RGB channels

numClasses = numel(classes);

network = deeplabv3plus(imageSize,numClasses,"resnet18");

%Balance Classes Using Class Weighting

imageFreq = tbl.PixelCount ./ tbl.ImagePixelCount;
classWeights = median(imageFreq) ./ imageFreq;

%Select Training Options

options = trainingOptions("sgdm",...
    LearnRateSchedule="piecewise",...
    LearnRateDropPeriod=6,...
```



```
LearnRateDropFactor=0.1,...
Momentum=0.9,...
InitialLearnRate=1e-2,...
L2Regularization=0.005,...
ValidationData=dsVal,...
MaxEpochs=18,...
MiniBatchSize=4,...
Shuffle="every-epoch",...
CheckpointPath=tempdir,...
VerboseFrequency=10,...
ValidationPatience=4);

%Start Training
doTraining = false;
if doTraining
    [net,info] =
        trainnet(dsTrain,network,@(Y,T) modelLoss(Y,T,classWeights),options);
end

% Test Network on One Image
%Employ of the pretrainNet
load('miRedEntrenadaT1num2.mat')
I = readimage(imdsTest,1);
C = semanticseg(I,net,Classes=classes)

cmap = camvidColorMap;
B = labeloverlay(I,C,Colormap=cmap,Transparency=0.4)

imshow(B)
hold on
pixelLabelColorbar(cmap, classes)

expectedResult = readimage(pxdsTest,1);
actual = uint8(C);
expected = uint8(expectedResult);
imshowpair(actual, expected)

iou = jaccard(C,expectedResult);
table(classes,iou)

%Evaluate Trained Network

pxdsResults = semanticseg(imdsTest,net, ...
```

```

Classes=classes, ...
MiniBatchSize=4, ...
WriteLocation=tmpdir, ...
Verbose=false);

metrics = evaluateSemanticSegmentation(pxdsResults,pxdsTest,Verbose=false);

metrics.DataSetMetrics

metrics.ClassMetrics

%Test on experimental data

% Path to the folder with real images and manual masks of the experiment:
testImgDir = 'C:\Users\mikio\Desktop\TestImages\PreprocessTestImagesRaw';
testMaskDir = 'C:\Users\mikio\Desktop\TestImages\PreprocessTestImagesSegmentation';

% Test file name
testFile = '001.png'; % use the correct picture

% 1. Read image and label photo with same name
I_test = imread(fullfile(testImgDir, testFile));
GT_test = imread(fullfile(testMaskDir, testFile));

%The block makes sure that your “ground truth”
(GT_test) always ends up as a categorical array of class labels,
whether you receive it in RGB format (colored image) or if you
already have it as an array of indices.
if ndims(GT_test) == 3
    % convert RGB-coded ground truth to numeric label indices
    GT_test_label = rgb2label(GT_test, cmap, classes); % Convierte RGB a etiquetas
else
    % 2D label matrix: convert numeric labels to a categorical array
    %GT_test_label = categorical(GT_test, 1:numel(classes), classes);
    GT_test_label = categorical( ...
        GT_test, ... % input matrix of label indices (1...N)
        1:numel(classes), ... % valid label values
        classes ); % names of each category
end

% 2.using raw image and network make prediction
C_test = semanticseg(I_test, net, Classes=classes);

```

```
% 3. show the prediction and manual mask (both on the original image)
figure;
subplot(1,2,1);
imshow(labeloverlay(I_test, C_test, Colormap=cmap, Transparency=0.4));
title('Net Prediction'); % Predicción red

subplot(1,2,2);
imshow(labeloverlay(I_test, GT_test_label, Colormap=cmap, Transparency=0.4));
title('Manual mask'); % Máscara manual

% 4. Calculate the accuracy per class (IOU)
iou_test = jaccard(C_test, GT_test_label);
T_test = table(classes(:), iou_test(:), 'VariableNames', {'Clase', 'IOU'});
disp('Accuracy per class (IOU) for the test image:');
disp(T_test);

% 5. Displays the average IOU over valid classes
fprintf('\nIOU average of classes: %.4f\n', mean(iou_test(~isnan(iou_test))));

%Supporting Functions

function labels = rgb2label(maskRGB, cmap, classNames)
labels = zeros(size(maskRGB,1), size(maskRGB,2));
for k = 1:numel(classNames)
    color = uint8(round(cmap(k,:) * 255));
    match = maskRGB(:,:,1) == color(1) & maskRGB(:,:,2) ==
        color(2) & maskRGB(:,:,3) == color(3);

    labels(match) = k;
end
labels = categorical(labels, 1:numel(classNames), classNames);
end
;

function labelIDs = camvidPixelLabelIDs()
labelIDs = { ...

    % "Pool Background"
    [
    160 196 189; ...
    ]
}
```

```
% "Black Cilinder"
[
141 195 141; ...
]

% "Silver Cilinder"
[
199 187 116; ...
]

% "Wooden Piece"
[
165 92 174; ...
]

% Granite cobblestone
[
197 197 197; ...
]
    };
end

function classes = getClassNames()
classes = [
    "PoolBackground"
    "BlackCilinder"
    "SilverCilinder"
    "Wooden Piece"
    "Granite"
];
end

function pixelLabelColorbar(cmap, classNames)
% Add a colorbar to the current axis. The colorbar is formatted
% to display the class names with the color.

colormap(gca,cmap)

% Add colorbar to current figure.
c = colorbar('peer', gca);

% Use class names for tick marks.
```

```
c.TickLabels = classNames;
numClasses = size(cmap,1);

% Center tick labels.
c.Ticks = 1/(numClasses*2):1/numClasses:1;

% Remove tick mark.
c.TickLength = 0;
end

function cmap = camvidColorMap()
% Define the colormap used by CamVid dataset.

cmap = [
    160 196 189    % Pool
    141 195 141    % Black Cilinder
    199 187 116    % Silver Cilinder
    165  92 174    % Wooden Piece
    197 197 197    % Granite
];

% Normalize between [0 1].
cmap = cmap ./ 255;
end

function [imdsTrain, imdsVal, imdsTest, pxdsTrain, pxdsVal, pxdsTest] =
partitionCamVidData(imds,pxds)
% Partition CamVid data by randomly selecting 60% of the data for training.
The rest is used for testing.

% Set initial random state for example reproducibility.
rng(0);
%numFiles = numpartitions(imds);
numFiles = numel(imds.Files);
shuffledIndices = randperm(numFiles);

% Use 60% of the images for training.
numTrain = round(0.60 * numFiles);
trainingIdx = shuffledIndices(1:numTrain);

% Use 20% of the images for validation
numVal = round(0.20 * numFiles);
valIdx = shuffledIndices(numTrain+1:numTrain+numVal);
```

```
% Use the rest for testing.
testIdx = shuffledIndices(numTrain+numVal+1:end);

% Create image datastores for training and test.
imdsTrain = subset(imds,trainingIdx);
imdsVal = subset(imds,valIdx);
imdsTest = subset(imds,testIdx);

% Create pixel label datastores for training and test.
pxdsTrain = subset(pxds,trainingIdx);
pxdsVal = subset(pxds,valIdx);
pxdsTest = subset(pxds,testIdx);
end

function data = augmentImageAndLabel(data, xTrans, yTrans)
% Augment images and pixel label images using random reflection and
% translation.

for i = 1:size(data,1)

    tform = randomAffine2d(...
        XReflection=true,...
        XTranslation=xTrans, ...
        YTranslation=yTrans);

    % Center the view at the center of image in the output space while
    % allowing translation to move the output image out of view.
    rout = affineOutputView(size(data{i,1}), tform, BoundsStyle='centerOutput');

    % Warp the image and pixel labels using the same transform.
    data{i,1} = imwarp(data{i,1}, tform, OutputView=rout);
    data{i,2} = imwarp(data{i,2}, tform, OutputView=rout);

end
end

function data = augmentImageAndLabel(data, xTrans, yTrans)
% Augment images and pixel label images using random reflection and
% translation.

for i = 1:size(data,1)
```

```
tform = randomAffine2d(...
    XReflection=true,...
    XTranslation=xTrans, ...
    YTranslation=yTrans);

% Center the view at the center of image in the output space while
% allowing translation to move the output image out of view.
rout = affineOutputView(size(data{i,1}), tform, BoundsStyle='centerOutput');

% Warp the image and pixel labels using the same transform.
data{i,1} = imwarp(data{i,1}, tform, OutputView=rout);
data{i,2} = imwarp(data{i,2}, tform, OutputView=rout);

end
end
```

E Appendix- Results

Each subsection presents the results obtained by training the DeepLab v3+ architecture with a ResNet-18 backbone. The figures are organized to display the raw image, the manually labeled ground truth, and the corresponding prediction. Within each subsection, the results are further divided according to the two experimental positions analyzed.

E.0.1 No Water Model - Real Test Predictions

This section presents the prediction results of the new system using the ResNet-based network under two training conditions: one designed to minimize overfitting, as described in the CNN section, and another where overfitting was intentionally allowed. Due to time constraints, a detailed comparison between both configurations could not be conducted; however, preliminary results from the overfitted network appeared to be promising.

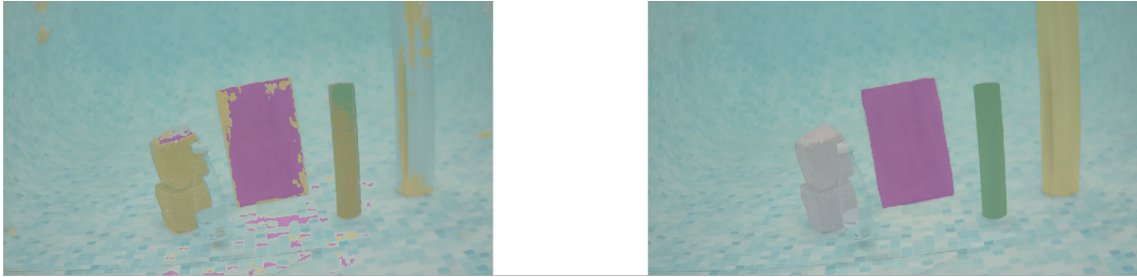


Figure E.1: Old Predictions for Baseline experimental images

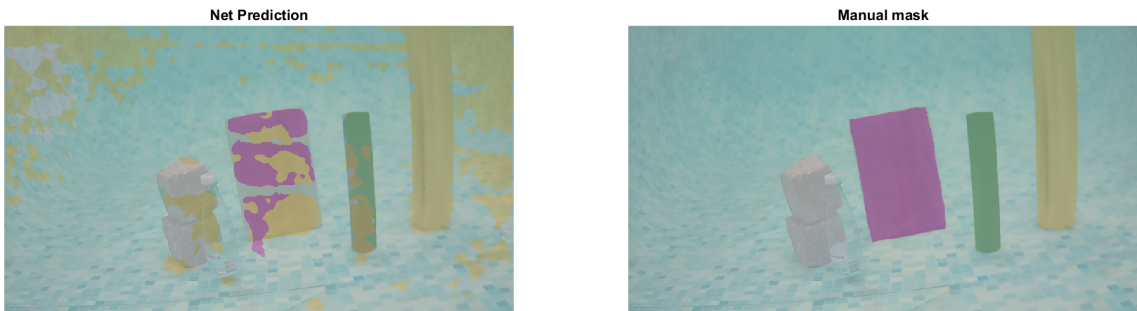


Figure E.2: New Predictions for Baseline experimental images

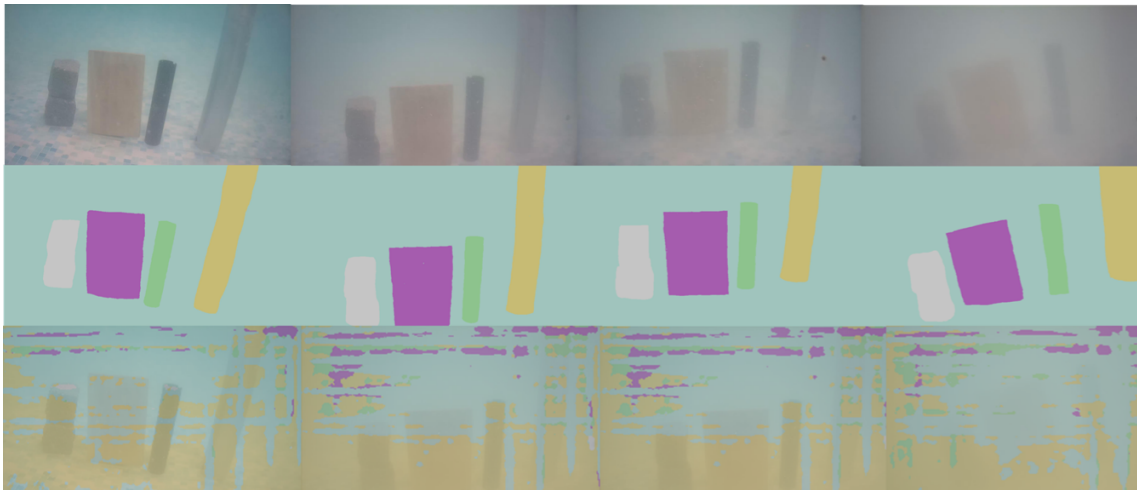


Figure E.3: Predictions for No Water Network-Position 1

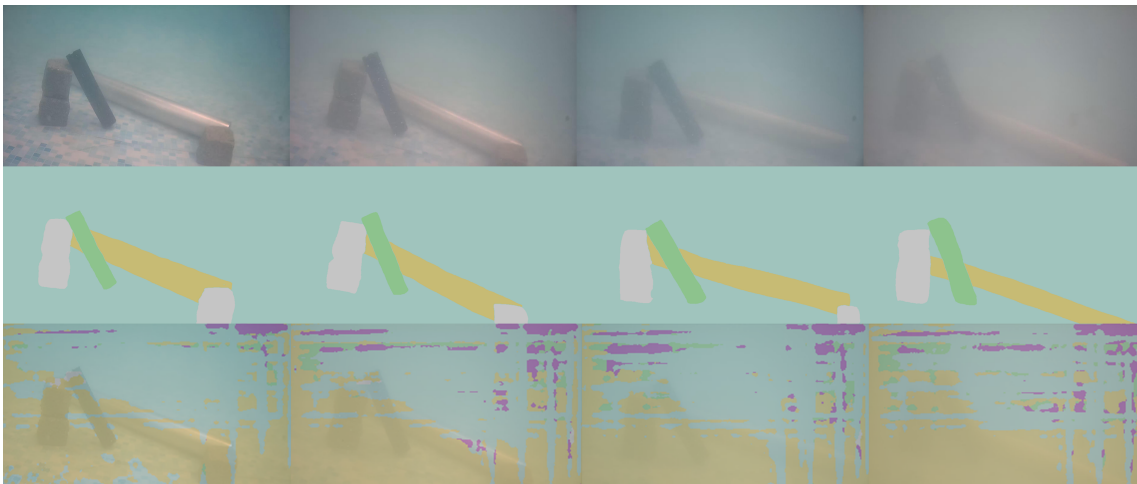


Figure E.4: Predictions for No Water Network-Position 2

E.0.2 Low Turbidity Model - Real Test Predictions



Figure E.5: Predictions for Low turbidity Network-Position 1

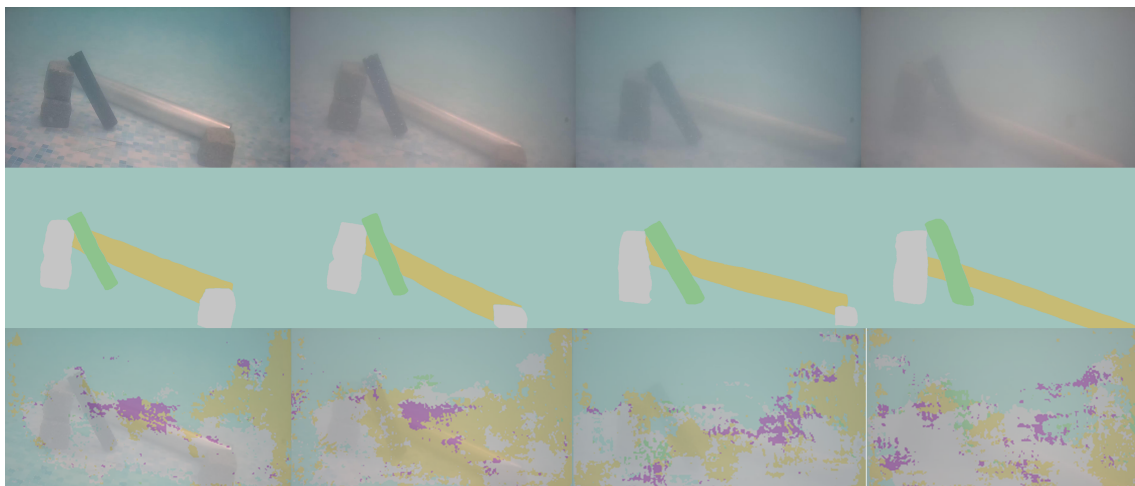


Figure E.6: Predictions for Low turbidity Network-Position 2

E.0.3 Medium Turbidity Model - Real Test Predictions



Figure E.7: Predictions for moderate turbidity Network-Position 1

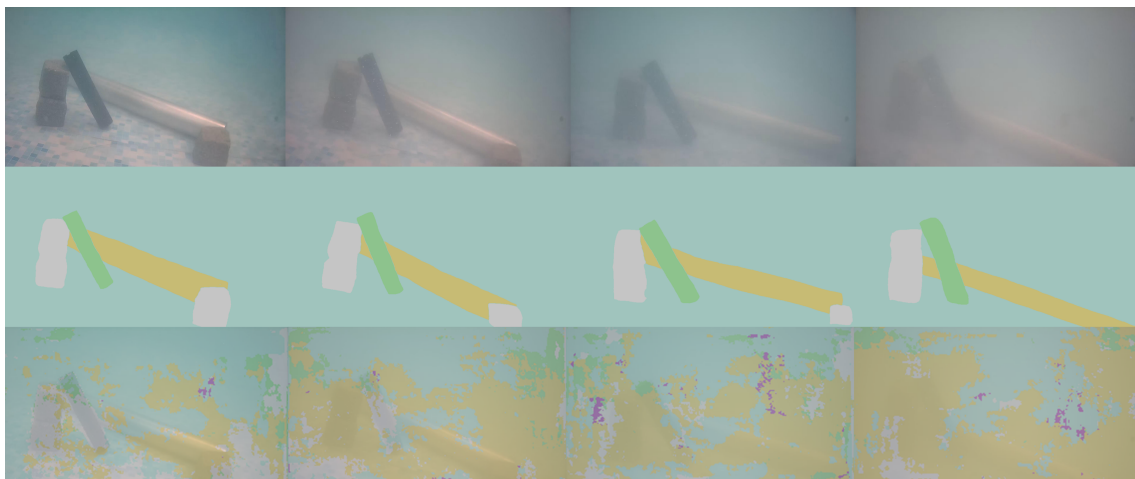


Figure E.8: Predictions for moderate turbidity Network-Position 2

E.0.4 High Turbidity Model - Real Test Predictions

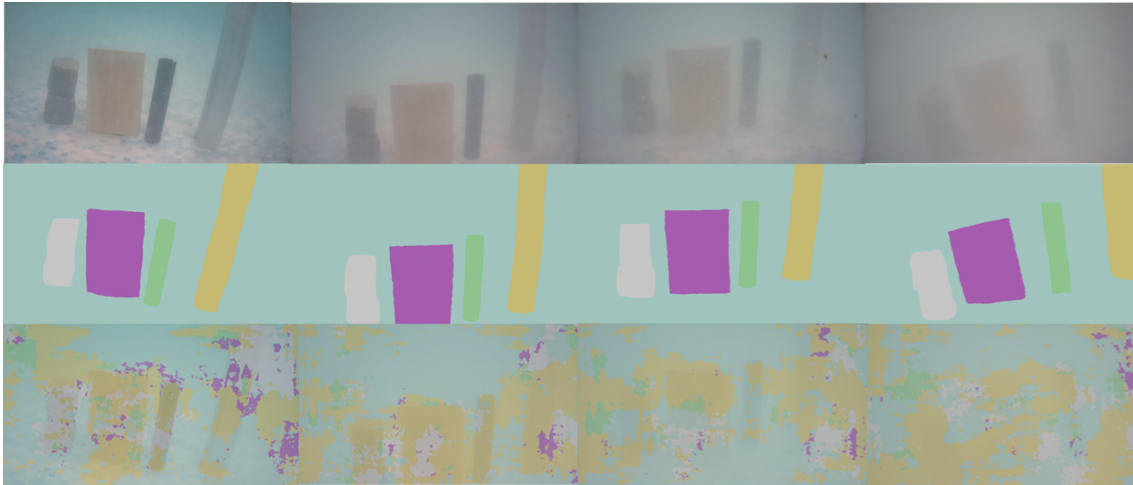


Figure E.9: Predictions for high turbidity Network-Position 1

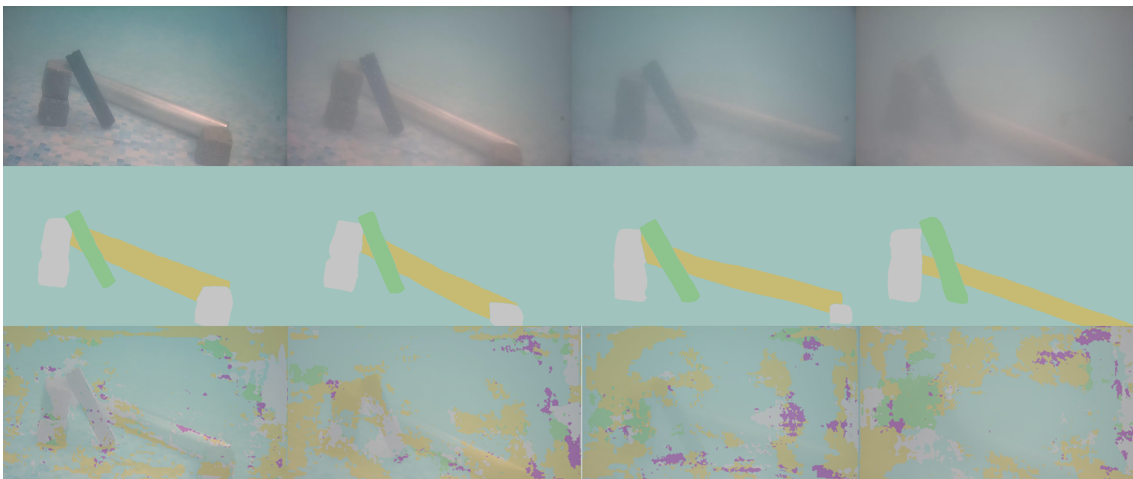


Figure E.10: Predictions for high turbidity Network-Position 2

E.0.5 All images Model - Real Test Predictions

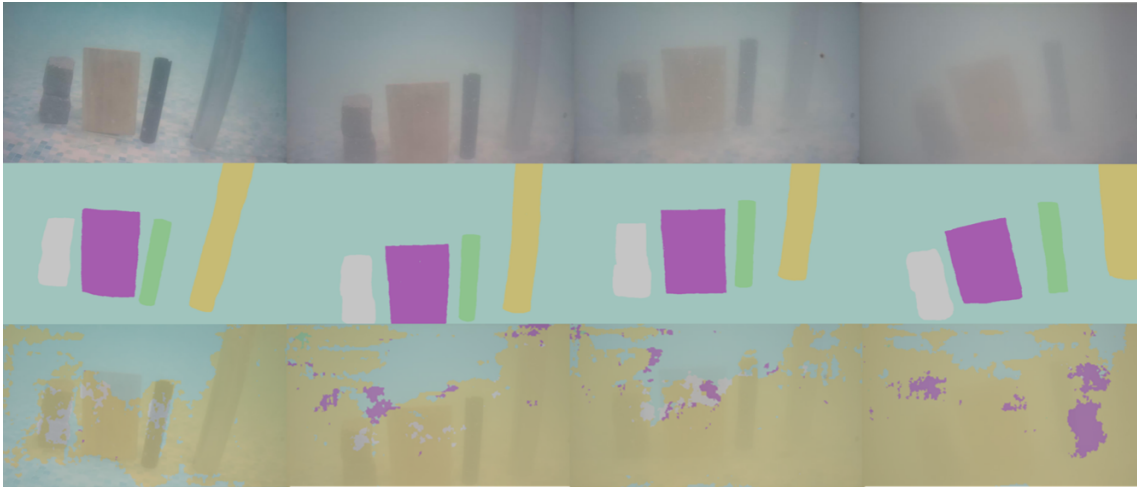


Figure E.11: Predictions for full data Network-Position 1

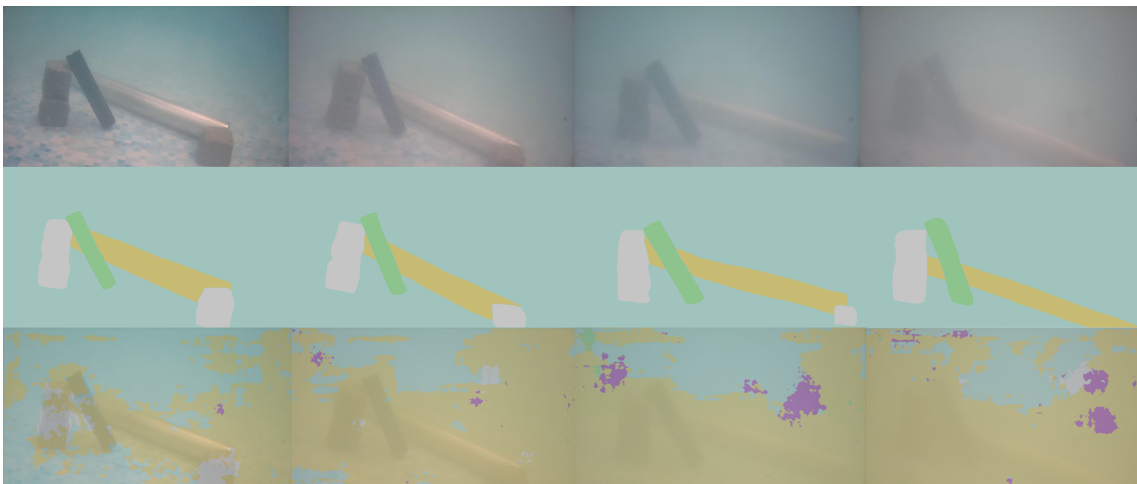


Figure E.12: Predictions for full data Network-Position 2

F Appendix - Frame Extraction Script

```
% Set the main working directory
workingDir = "C:\Users\mikio\Desktop\ImagenPrueba\Extraerframes";

% Create the main folder
mkdir(workingDir)

% Create a subfolder to store the extracted images
mkdir(workingDir,"images8")

% Load the video file
shuttleVideo = VideoReader("P2.T4.mp4");

% Initialize frame counter
i = 1;

% Loop through all frames of the video
while hasFrame(shuttleVideo)
    % Read current frame
    img = readFrame(shuttleVideo);

    % Create a filename with padded numbering (e.g., 001.jpg)
    filename = sprintf("%03d",i)+".jpg";

    % Build the full path to save the image
    fullname = fullfile(workingDir,"images8",filename);

    % Save the frame as a JPEG image
    imwrite(img,fullname)

    % Increment frame counter
    i = i+1;
end
```