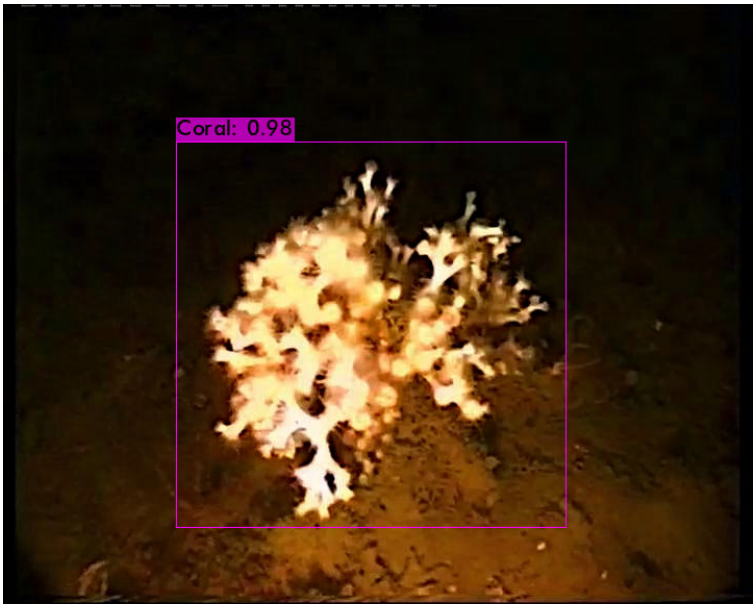

Ocean Exploration with Artificial Intelligence

A Detection System for Deepwater Corals Using Generative Adversarial Networks

Master's thesis in Computer science and engineering

Sarah Al-Khateeb, Lisa Bodlak

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2021

Master's thesis 2021

Ocean Exploration with Artificial Intelligence

A Detection System for Deepwater Corals Using Generative Adversarial Networks

Sarah Al-Khateeb
Lisa Bodlak

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2021

Ocean Exploration with Artificial Intelligence
A Detection System for Deepwater Corals Using Generative Adversarial Networks
Sarah Al-Khateeb, Lisa Bodlak

Supervisor: Matthias Obst, Marine Sciences
Examiner: Peter Damaschke, Computer Science and Engineering

Master's Thesis 2021
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A Deepwater Coral detected by our proposed system.

Typeset in LaTeX
Gothenburg, Sweden 2021

Ocean Exploration with Artificial Intelligence
A Detection System for Deepwater Corals Using Generative Adversarial Networks
Sarah Al-Khateeb, Lisa Bodlak
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Large and diverse data is crucial to train object detection systems properly and achieve satisfactory prediction performance. However, in some areas, such as marine science, gathering sufficient data is challenging and sometimes even infeasible. Working with limited data can result in overfitting and poor performance. Furthermore, underwater images suffer from various problems, like varying quality, which have to be considered. Therefore, alternative means need to be used to increase and enhance the data to facilitate marine scientists' work.
In this thesis, we explore building a more robust system to improve the detection accuracy for deepwater corals and analyze underwater movies under different conditions. We experiment with several Generative Adversarial Networks (GANs) to enhance and increase the training data. Our final system comprises two steps: Image Augmentation using StyleGAN2 combined with the augmentation strategy DiffAugment, and Object Detection using YOLOv4.
The results indicate that generating realistic synthetic data combined with an advanced detector could provide marine scientists with the tool they need to extract species occurrence information from underwater movies. Our proposed system shows increased performance in different domains compared to prior work and the potential to overcome the limited data issue.

Keywords: computer science, deep learning, generative adversarial networks, data augmentation, object detection, underwater image, computer vision.

Acknowledgements

We want to express our gratitude to our Supervisor Matthias Obst and the whole KSO team, especially Victor Anton and Jannes Germishuys, for their guidance and helpful feedback throughout this thesis. We also want to thank our examiner Peter Damaschke for the valuable feedback for the midterm report. Last but not least, we would like to acknowledge MMT Sweden AB for sharing data with us to use in this research.

Sarah Al-Khateeb, Lisa Bodlak, Gothenburg, June 2021

Contents

List of Figures

List of Tables

1
Introduction

The distribution of species in the ocean is to a large extent unknown [2]. The Ocean Biodiversity Information System (OBIS) has only collected data about 5% of all species in the ocean. In order to explore the ocean, automated systems for research are needed. Remotely Operated Vehicles (ROVs) made it possible to collect a large amount of marine footage, but the current manual analysis methods are too time-consuming to exploit the full potential of the available data [3]. Therefore, ideas have been generated towards finding a fast and efficient solution by using machine learning algorithms.

Humans are able to easily recognize and locate objects of interest in an image at once. However, performing and passing this intelligence to computers is more complicated. A field that deals with this matter is called computer vision, which teaches computers how to process images. Several machine learning techniques are available to be used on images in the computer vision field. One of these techniques is image classification [4], which is used to classify a whole image into one category. Object detection [5] is another technique that deals with locating and classifying multiple objects in a single image.

When analyzing marine footage, there is often more than one object of interest in a frame; this is where object detection comes into hand. However, the challenge with subsea data is the unexpected environment under the sea and the quality of the captured images at a given moment. Some other challenges include noise due to small sea particles and color distortion. Moreover, variations in the undersea environment, limited light conditions, and different locations can affect the data analysis [6].

1.1 Problem

Koster Seafloor Observatory (KSO) [7] is a system that combines citizen science[1] and machine learning for automated analysis of subsea movies. The system uses citizen science to annotate samples from target data that needs to be analyzed. The annotations with the highest agreement are then used to train an object detection system, which can be used to recognize objects in the target data and extract abundance information.

The study area of KSO is the Kosterhavet National Park, which is under active

[1] Citizen Science describes the contribution of the public to a research project.

protection since 2009. ROVs have been used in the last 30 years to monitor this area. The collected movies can be used to study how various pressures (e.g. trawling, eutrophication) influence the species in this area and which positive effects the establishment of the national park had.

Up to now, the system has been tested on detecting one category, namely the deepwater coral species *Desmophyllum pertusum (Linnaeus, 1758)*. To do so, a sample of 60 one-hour movies has been selected from the collected movies of the Kosterhavet National Park and annotated with the help of the citizen science module.

In the citizen science module, the one-hour movies were first split into 10-second clips, of which 5702 clips were randomly selected. Then citizen scientists were presented with a clip where they identified the species and recorded the time the species first fully appeared. Each clip got annotated by eight different persons, and only clips with an agreement of more than 80% were retained. The frames where the species first fully appeared got extracted and presented to another five citizen scientists, who had the task to locate the species by drawing a rectangle around them. After filtering the frames for matching criteria, the overlap area was used as the final annotation, which resulted in a labeled dataset of 409 frames.

To increase the labeled data for training, the KSO team used a frame tracker [8] to fill in the intermediate frames by tracking the identified objects. Tracking the objects was done for additional ten frames (0.5 seconds of footage). The resulting dataset included 4499 frames. Furthermore, simple pre-processing was done to remove background distortion. Finally, an object detector (YOLOv3 [9]) was trained and used to compute the spatial distribution of cold water coral in the Kosterhavet National Park.

The pre-processing pipeline used by the KSO team was a starting point to speed up the annotation process. However, increasing the data by using a tracker might not add much diversity to the dataset. This is because, if the camera did not move and capture the species from another angle, it would result in adding almost the same image multiple times to the dataset. This brings the risk of introducing bias into the training dataset, and therefore, the model would not generalize well to unseen data. Therefore, we will work together with the KSO team to propose a more robust model that allows the KSO team to analyze movie data in new environments and conditions.

1.2 Goals

KSO has been tested to extract ecological data with a YOLOv3 model for one species, the deepwater corals. The results seemed promising but with a limitation that it could not operate well in unseen data. This research aims to propose a robust system that overcomes the overfitting problem and can generalize to different subsea environments.

An essential part of computer vision is to have a large and diverse dataset to train models properly and achieve desired outcomes [10]. However, obtaining large labeled

data can be a complex and expensive task. Preparing and annotating the data manually to be used in a specific algorithm is time-consuming. In addition, some species are rare in the ocean, such as deepwater corals, so sufficient data is hard to obtain. Therefore, the annotated training data from the citizen science module in KSO is relatively small; 409 annotated frames were used in the pilot run for the deepwater corals. To overcome the problem of overfitting, we need to fall back on other methods to increase the training data. The second challenge is that underwater images suffer from bad quality due to color distortion, low contrast, and blur, which results in weak performance [6]. To address this problem, appropriate pre-processing steps have to be taken.

Different approaches can be used to improve and increase the training data. Generative Adversarial Networks (GANs) showed an increase in performance compared to classical augmentation, for example in the field of medical image classification [11]. Classical Augmentation applies transformations and so adds variety to a certain extent. However, it can not introduce enough diversity to properly train deep learning models. Therefore, we decided to work with GANs. Different GANs have been proposed in the last years, and we aim to test several appropriate versions to create a suitable pipeline that can enhance and increase the data. After obtaining a representative dataset, we need to train an algorithm and test it on unseen data to evaluate this dataset's reliability and make further improvements. Different object detection systems exist; we decided to use YOLOv4. This choice is based on a compromise between choosing a strong detector with good performance, but which also requires little computing power, so that marine scientists can easily integrate this system into their work. Finally, during testing we want to compare our model to the KSO model (YOLOv3) and answer the following three research questions:

1. Can our proposed system overcome the overfitting issue on data from the Kosterhavet National Park with the same environmental settings?
2. Can our proposed system generalize to different environments inside the Kosterhavet National Park?
3. Can our proposed system generalize to different environments outside the Kosterhavet National Park?

1.3 Related Work

KSO combines data management, citizen science, and machine learning in one system to make it easy for researchers to store, process, and analyze their data. Although other systems that use machine learning to analyze the ocean have been presented, KSO is the first one combining these three modules.

One recent work is done by Boulais et al. [12]. They are building FathomNet, an underwater image training set for the development of machine learning algorithms. The dataset consists of more than $80,000$ images including 233 different classes. Even though their experiments have shown promising results, they argue that they need an even bigger dataset to overcome the data shift problem and successfully train a global system that can be used for research in the ocean. In contrast to

FathomNet, the idea of KSO is to develop custom training data for a research problem at hand, and hence the algorithm is more likely to not suffer as much from out-of-distribution testing data.

Another recently presented system is done by Ditria et al. [13]. They used a Mask Region-Based Convolutional Neural Network (Mask R-CNN) [14] to analyze fish abundance. While they concluded that training on data with different environmental conditions improves the analysis, our approach uses data denoising and augmentation techniques to improve the robustness of the model.

Lopez-Vazquez et al. [15] introduced a system for underwater animal detection using segmentation and classification methods. They applied classical pre-processing, such as the Contrast Limited Adaptative Histogram Equalization (CLAHE) [16] to enhance the contrast. To increase their data, they make use of traditional augmentation techniques, such as flipping and rotation. Furthermore, Song et al. [17] proposed a Mask R-CNN recognition system for underwater species using small training data. Besides traditional augmentation techniques like flipping and adding noise, they also used SinGAN [18], a Generative Adversarial Network that can generate new images from a single image to increase the training data. Furthermore, they used a Multi-Scale Retinex with Color Restoration (MSRCR) algorithm as a pre-processing step in their pipeline to enhance the images. However, SinGAN did not perform well, and most of the generated images were not realistic, which resulted in only adding a few manually picked images to the original ones. In contrast to the two systems mentioned above, our approach is to build an object detection pipeline that uses Generative Adversarial Networks (GANs) for image enhancement and augmentation.

1.4 Ethical Considerations

In this research, we will be working with underwater images that do not contain any personal data. However, some aspects need to be considered regarding the development phase and the investigated outcome.

1.4.1 Ethical aspects of the development phase

The underwater images we are using during the development of our model include geographic information of depth. This information is not allowed to be published according to the Swedish law [19]. This means that we will check all the time that we exclude depth information from the data. We will ensure this by cross-reviewing all data, documentation, and reports at regular intervals.

1.4.2 Ethical aspects of the research outcome

The goal of our research is to integrate a better machine learning module into the KSO system, which can help in accurately detecting species in different environments and provide marine scientists with more data for further analysis. This will allow researchers to extract information for many species and help monitor and take

appropriate actions in the future. However, when such information falls into the wrong hands, it can be used for evil acts such as poaching. To mitigate this, marine researchers need to be careful about publishing geographical information on species abundance.

1.5 Roadmap

The rest of this thesis is organized as follows: Chapter 2 introduces the theoretical knowledge for the algorithms, techniques, and metrics used in this thesis. Chapter 3 contains information about the data, a detailed description of the methods used, and an outline of how the training, evaluation, and testing is applied in every step of our pipeline: image enhancement, image augmentation, and object detection. Chapter 4 includes the results and discussion for every step and Chapter 5 presents our conclusion. Finally, Chapter 6 discusses different ideas for future work.

2
Theory

2.1 Generative Adversarial Networks

Generative Adversial Networks (GANs) were firstly introduced in 2014 by Goodfellow et al. [20]. The idea of GANs is to learn the distribution of some data in a two-player game without explicitly modeling the density. The network consists of two fully connected feed-forward networks, the generator G and the discriminator D. The generator G produces fake data from input noise z and tries to trick the discriminator D. The discriminator D is a binary classifier with the task to distinguish real data x from fake data $G(z)$. The architecture of a GAN is demonstrated in [21, Figure 2.1]. Table 2.1 gives an overview of the symbols used to introduce the theory of GANs.

Symbol	**Description**
$\mathbb{P}_r$	Distribution of the real data x.
$\mathbb{P}_\theta$	Distribution of the fake data $G(z)$ generated by the Generator G.
$\mathbb{P}_z$	Prior Distribution of the noise z.
x	Real data sample from $\mathbb{P}_r$.
z	Noise sample from $\mathbb{P}_z$.
D	The Discriminator Network: formally defined as the function $D(x; w)$
G	The Generator Network: formally defined as the function $G(z; \theta)$.
w	The parameters of $D(x; w)$, i.e. the weights of the network that are adjusted during the training.
θ	The parameters of $G(z; \theta)$, i.e. the weights of the network that are adjusted during the training.
$D(x)$	Scalar value between 0 and 1 that represents the probability of the real data sample x belonging to the real data.
$G(z)$	The Generator's output: fake data sample.
$D(G(z))$	Scalar value between 0 and 1 that represents the probability of the fake data sample $G(z)$ belonging to the real data.

Table 2.1: List of symbols with corresponding description in the GAN setting.

Formally, Goodfellow et al. [20] define the two functions $D(x; w)$ and $G(z; \theta)$, represented by two fully connected feed-forward networks with learnable parameters w and θ respectively. $G(z; \theta)$ maps noise z, sampled from a noise prior $\mathbb{P}_z$, to the data space and tries to approximate the real data distribution $\mathbb{P}_r$ with the generator's distribution $\mathbb{P}_\theta$. The input space of $D(x; w)$ is the union of real data $x \sim \mathbb{P}_r$ and fake

data $G(z) \sim \mathbb{P}_\theta$. $D(x; w)$ outputs for each data point individually the probability of belonging to the real data, denoted by $D(x)$ and $D(G(z)) \in [0, 1]$ respectively.

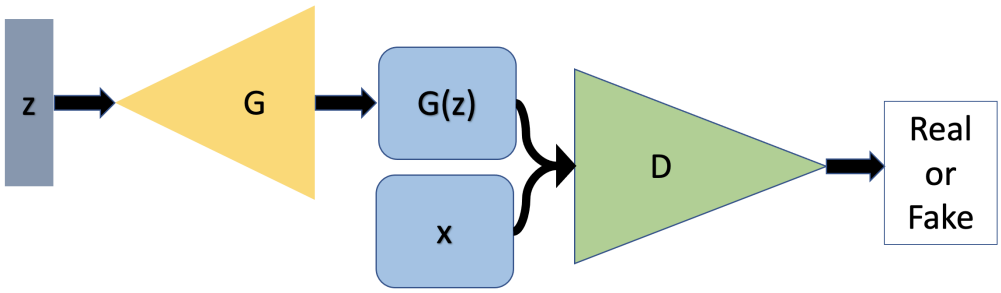

Figure 2.1: Architecture of GANs.

In order to train a network, or in this case two networks, the problem has to be translated into a loss function. The loss function summarizes the difference between predictions and ground truth, which we try to minimize. The discriminator is a binary classifier, so it's loss function can be modeled with cross-entropy [22]. Since the generator's goal is the opposite of the discriminator, it's loss function is the negative of the discriminator's loss. Consequently the two models formulate a minimax game with the objective function [20, Equation 2.1]

$$\min_G \max_D \mathbb{E}_{x \sim \mathbb{P}_r}[\log D(x)] + \mathbb{E}_{z \sim \mathbb{P}_z}[\log(1 - D(G(z)))]. \tag{2.1}$$

The discriminator tries to catch the generator by maximising the probability of assigning the correct class such that $D(x) \approx 1$ and $D(G(z)) \approx 0$. However, the generator wants to minimize the discriminator being correct such that $D(G(z)) \approx 1$. The network is jointly trained until D cannot differentiate anymore between real and fake data, and hence G has successfully learned the representation of real data. The training procedure to optimize Equation 2.1 is shown in [20, Algorithm 1]. In each training iteration, the discriminator's weights w gets first updated for k steps before the generator's weights θ are updated for one step.

Optimizing the generator objective does not work well in practice since the gradient is likely to vanish in the training procedure's early stages. A solution to this is to maximize the likelihood $\log(D(G(z)))$ of the discriminator being wrong instead of minimising the likelihood of the discriminator being right i.e. $\log(1 - D(G(z)))$. This gives a higher gradient in the early learning stages and does not change the dynamics of G and D.

Algorithm 1: Minibatch stochastic gradient descent training of GANs. Source: [20]

for *number of training iterations* **do**
  for *k steps* **do**
    1. Sample minibatch of m noise samples $z^{(1)}, ..., z^{(m)}$
    2. Sample minibatch of m examples $x^{(1)}, ..., x^{(m)}$
    3. Stochastic Gradient Ascent on discriminator D:
       $\nabla_w \frac{1}{m} \sum_{i=1}^{m} [\log D(x^{(i)})] + \log(1 - D(G(z^{(i)})))]$
  end
  1. Sample minibatch of m noise samples $z^{(1)}, ..., z^{(m)}$
  2. Stochastic Gradient Ascent on generator G (improved objective):
     $\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} \log(D(G(z^{(i)})))$
end

Once the model is trained, the generator G allows the generation of synthetic data from the data distribution. However, a problem with GANs is that they can get into mode collapse, meaning that G maps many noise vectors z to the same image x, resulting in a lack of diversity in the generated samples. Different ideas have been generated to address this problem, including changes in the architecture and the objective function. Three of the derived GANs, Deep Convolution GANs, Wasserstein GANs, and StyleGANs, are described in section 2.1.1, section 2.1.2, and section 2.1.3, respectively. Section 2.1.4 shows an application of GANs besides generating synthetic data. Finally, section 2.1.5 describes how data augmentation can be applied to GANs.

2.1.1 Deep Convolutional GAN

In 2016 Radford et al. proposed a new framework called Deep Convolutional Generative Networks (DCGN) [23]. They changed the generator and discriminator from fully connected layers to convolutional layers, which resulted in more stable training and the possibility to train deeper networks at a higher resolution. Furthermore, they used strided convolutional and strided convolutional transpose layers instead of deterministic pooling functions in the discriminator and generator to force the model to learn its own down- and upsampling, respectively. Finally, they used Batch Normalization to address the problem of mode collapse and unstable training and changed the activation functions resulting in quicker learning of the model.

The generator network is shown in [23, Figure 2.2]: an upsampling network with strided convolutional transpose layers and batch norm layers. Rectified Linear Unit (ReLU) activation is used on each layer except on the output layer, which uses the Hyperbolic Tangent (Tanh) function. The discriminator network compromises batch norm layers, strided convolutional layers, LeakyReLU layers, and the output layer uses a sigmoid activation.

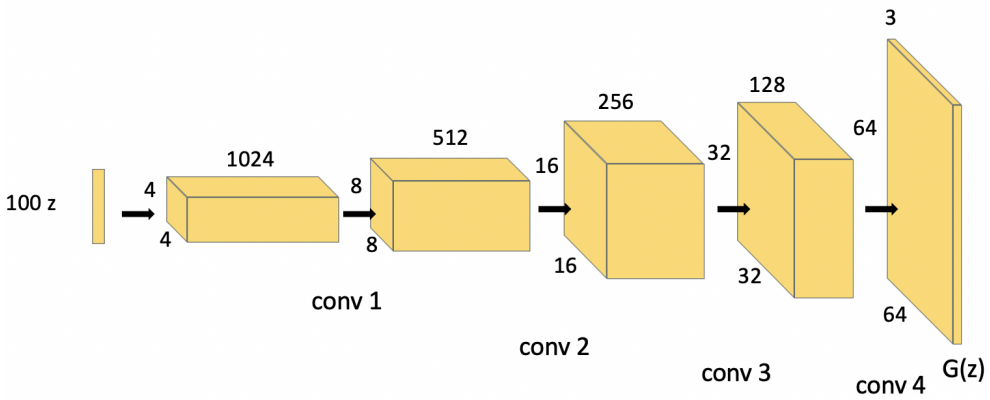

Figure 2.2: Architecture of the DCGAN Generator.

2.1.2 Wasserstein GAN

In order to improve the training stability and overcome the mode collapse issue in GANs, Arjovsky et al. introduced an extension to GANs named Wasserstein GAN (WGAN) [24]. The main idea in WGAN is to effectively measure the distance between the predicted distribution $\mathbb{P}_\theta$ and the real distribution $\mathbb{P}_r$ since choosing how to measure the distance might affect the convergence of the model.

With the Jensen–Shannon divergence used in the original GAN, when the generated distribution is very different from the real data distribution, the generator will learn nothing since the discriminator can easily distinguish between real and fake data and fail to provide useful information to the generator, in other words, the generator gradient vanishes. Arjovsky et al. introduced a new objective function [24, Equation 2.2] using the Wasserstein distance,

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_\theta)} \mathbb{E}_{(x,y)\sim\gamma}[\|x - y\|], \tag{2.2}$$

where γ is from the set of all joint distributions $\Pi(\mathbb{P}_r, \mathbb{P}_\theta)$ with marginals $\mathbb{P}_r$ and $\mathbb{P}_\theta$. Intuitively, the distance measures the cost of the optimal transport plan used to transform the distribution $\mathbb{P}_\theta$ into $\mathbb{P}_r$.

Computing the infimum in Equation 2.2 is hard, therefore, the authors used the Kantorovich-Rubinstein duality [25] to write the distance as shown in [24, Equation 2.3].

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \frac{1}{K} \sup_{\|F\|_L \leq K} \mathbb{E}_{x\sim\mathbb{P}_r}[F(x)] - \mathbb{E}_{x\sim\mathbb{P}_\theta}[F(x)]. \tag{2.3}$$

Here, the supremum is taken over all K-Lipschitz[1] functions $F : X \to \mathbb{R}$ with X denoted as the data space.

[1]For a function F to be K-Lipschitz the gradient of F has to be less than or equal K.

This problem can be written as shown in [24, Equation 2.4].

$$\max_{w \in W} \mathbb{E}_{x \sim \mathbb{P}_r}[F(x; w)] - \mathbb{E}_{x \sim \mathbb{P}_\theta}[F(G(z; \theta); w)]. \tag{2.4}$$

Here, $F(x; w)_{w \in W}$ is a parameterized family of K-Lipschitz functions for a given K. Solving this problem will yield $W(\mathbb{P}_r, \mathbb{P}_\theta)$ up to a multiplicative constant.

The WGAN training process is described in [24, Algorithm 2]. In this process, the discriminator D from the original GAN has been replaced with the network F, called "critic", which is used to learn the optimal K-Lipschitz function to solve the problem in Equation 2.4 instead of classifying an image as real or fake. To guarantee that the critic is a K-Lipschitz function, Arjovsky et al. use weight clipping, a method to restrict the critic's weights to be within a specific range to enforce the K-Lipschitz constraint. Moreover, they used RMSProp with a small learning rate and no momentum for stable training.

The new objective function optimizes the GAN training gradually. The Wasserstein distance has a smoother gradient that allows continuous learning until the critic achieves optimality, then the generator model is trained using the gradients from the optimal critic. In this case, the balancing between generator and discriminator is no longer needed since the generator will be trained once the critic has high-quality gradients.

Algorithm 2: WGAN Algorithm. Source: [24]

Variables: G, the generator. F, the critic.
Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.
Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.
while θ *has not converged* **do**
  for $t = 0,\ldots,n_{critic}$ **do**
    1. Sample $\{x^{(i)}\}_{i=1}^m$ a batch from the real data.
    2. Sample $\{z^{(i)}\}_{i=1}^m$ a batch of noise samples.
    3. $\nabla_w \leftarrow \nabla_w[\frac{1}{m}\sum_{i=1}^m F(x^{(i)}) - \frac{1}{m}\sum_{i=1}^m F(G(z^{(i)}))]$
    4. $w \leftarrow w + \alpha \cdot RMSProp(w, \nabla_w)$
    5. $w \leftarrow clip(w, -c, c)$
  end
  1. Sample $\{z^{(i)}\}_{i=1}^m$ a batch of noise samples.
  2. $\nabla_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^m F(G(z^{(i)}))$
  3. $\theta \leftarrow \theta - \alpha \cdot RMSProp(\theta, \nabla_\theta)$
end

However, according to the authors, the weight clipping method is not the best way to enforce the Lipschitz constraint, and tuning the clipping parameter c can be tricky. With a large c, we can face the issue of exploding gradients, while a smaller c might result in vanishing gradients and hence, affect the model's capacity to create complex functions. As a solution for this, an improved version of WGAN was introduced by Gulrajani et al. called Wasserstein GAN with Gradient Penalty (WGAN-GP) [26].

Instead of using weight clipping, WGAN-GP uses gradient penalty to penalize the network when the gradient of F shifts from $K = 1$ for points $\hat{x}$ interpolated between the real and generated data. The new objective is shown in [26, Equation 2.5].

$$\mathcal{L} = \underbrace{\mathbb{E}_{x\sim\mathbb{P}_r}[F(x)] - \mathbb{E}_{x\sim\mathbb{P}_\theta}[F(x)]}_{\textbf{WGAN critic loss}} + \underbrace{\lambda\mathbb{E}_{\hat{x}\sim\mathbb{P}_{\hat{x}}}[(\|\nabla_{\hat{x}}F(\hat{x})\|_2 - 1)^2]}_{\textbf{Gradient penalty}}. \tag{2.5}$$

Moreover, batch normalization was dropped from the critic since it affected the effectiveness of this approach. Batch normalization considers a batch of images, which conflicts with the gradient penalty, penalizing each gradient independently. The new training process is demonstrated in [26, Algorithm 3].

Algorithm 3: WGAN Algorithm with gradient penalty (WGAN-GP). Source: [26]

Variables: G, the generator. F, the critic.
Require: : λ, the gradient penalty coefficient. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration. α, β_1, β_2, the Adam hyperparameters[a].
Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.
while θ *has not converged* **do**
  for $t = 0,\ldots,n_{critic}$ **do**
    for $i = 1, ..., m$ **do**
      1. Sample x from real data, z from noise prior, $\epsilon \sim U[0, 1]$.
      2. $\tilde{x} \leftarrow G(z)$
      3. $\hat{x} \leftarrow \epsilon x + (1 - \epsilon)\tilde{x}$
      4. $\mathcal{L}^{(i)} \leftarrow F(\tilde{x}) - F(x) + \lambda(\|\nabla_{\hat{x}}F(\hat{x})\|_2 - 1)^2$
    end
    $w \leftarrow Adam(\nabla_w \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}^{(i)}, w, \alpha, \beta_1, \beta_2)$
  end
  1. Sample $\{z^{(i)}\}_{i=1}^{m}$ a batch of noise samples.
  2. $\theta \leftarrow Adam(\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} -F(G(z)), \theta, \alpha, \beta_1, \beta_2)$
end

[a] Adam [27] is an algorithm for stochastic optimization of a function with the core idea of computing adaptive learning rates based on the estimates of the first two moments of the gradient. The hyperparameter α is the learning rate and the hyperparameters β_1 and β_2 regulate the exponential decay rates of the moving averages, which estimate the first two moments of the gradient.

2.1.3 StyleGAN

The aforementioned extensions of GANs, like WGAN and DCGAN, can produce synthetic images but of small size. Karras et al. [28] introduced a state-of-the-art GAN, called StyleGAN, which can generate high-resolution images and uses a redesigned generator network to control the properties of generated images better.

StyleGAN builds up on Progressive GAN [29], which showed impressive results in generating high-resolution output. The idea of Progressive GAN is to start training with low-resolution images and add more layers progressively during the training

process to both the generator and discriminator like demonstrated in (a) in [28, Figure 2.3]. At the beginning of the training, the generator only consists of the first block, producing images of size 4×4. After some time, more layers are smoothly faded in, which are presented with the second block. The benefit of progressively growing networks is the stabilized training process in early stages, since the generator does not have to learn the mapping from noise to high resolution straight away, but rather in small steps. Furthermore, it reduces the training time, making it feasible to produce high-resolution synthetic images.

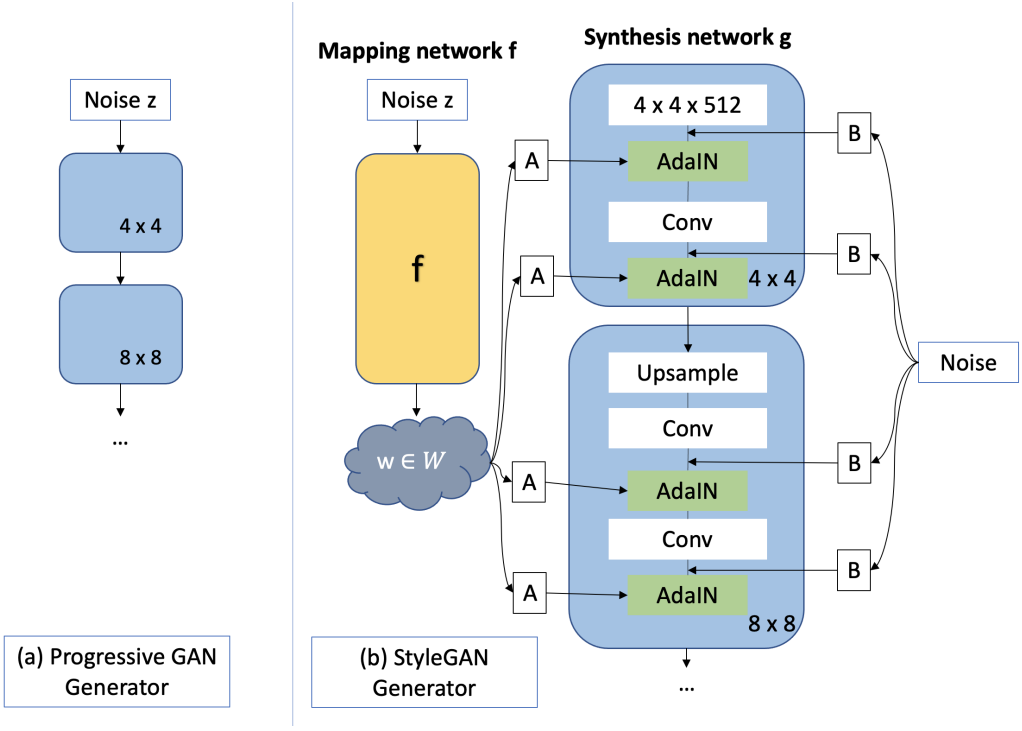

Figure 2.3: Comparison of the generator used in (a) Progressive GAN and (b) StyleGAN.

As demonstrated in Figure 2.3, the main contribution of StyleGAN was to redesign the generator architecture in Progressive GAN, which we will now explain further. In a vanilla GAN, like ProgressiveGAN, noise z is fed to the generator through an input layer. However, in StyleGAN, this input layer is omitted, and a mapping network with input z is employed, which outputs a latent vector $w \in W$, see Figure 2.3 (b) on the left. The mapping network f includes 8 fully-connected layers, which takes a 512 dimension noise vector z and transforms it into a 512 dimension intermediate latent vector $w \in W$. A learned affine transformation A is then applied to vector w to obtain a style vector. The style vector is transformed using adaptive instance normalization (AdaIN) [30] and integrated into each block after each convolutional layer of the generator network g, see Figure 2.3 (b) on the right.
Since the generator network g no longer takes a noise vector as input, the synthesis network g takes a learned constant tensor with dimension $4 \times 4 \times 512$ as input. Furthermore, the authors introduced an additional noise input B, which is added to the feature maps before the AdaIN operation to add a stochastic variation in an

image.

The proposed StyleGAN achieved state-of-the-art results. However, some generated images contained artifacts (water drops). Therefore the authors revised and redesigned StyleGAN and presented a new model, called StyleGAN2 [31]. The two main changes included redesigning the normalization used in the generator to overcome the artifacts issue and changing from using progressively growing during training to skip connections and residual networks instead.

2.1.4 CycleGAN

While the previous GANs focused on solving the original GAN limitations, Zhu et al. [32] proposed CycleGAN, an unsupervised approach to the paired image-to-image translation problem. Image-to-image translation requires training data of paired images to perform the translation from an input domain to an output domain. However, CycleGAN can learn the style of a group of images and transfer it to another group of images without pairing.

CycleGAN comprises two generator models G, F, and two discriminator models D_X, D_Y that are trained jointly. The generator G takes images from domain X and converts them to images in the domain Y, the second generator F takes images from domain Y and converts them back to domain X. Each generator has a discriminator that is used to distinguish real images from fake images, see (a) in [32, Figure 2.4].

The CycleGAN objective function has two components: adversarial losses and a cycle consistency loss. The adversarial losses, which match the distribution of generated and real images, were not sufficient to guarantee the consistency between G and F since they are mainly used for image generation and not translation tasks. However, the cycle consistency loss addresses this issue. The idea of introducing this loss is that when we transfer an image to another domain and back again, we should land on the same image. It enforces that $F(G(X)) \approx X$ and $G(F(Y)) \approx Y$, see (b) and (c) in [32, Figure 2.4]. The full objective including both loss terms is defined in [32, Equation 2.6].

$$\mathcal{L} = \underbrace{\mathcal{L}_{GAN}(G, D_Y, X, Y) + \mathcal{L}_{GAN}(F, D_X, Y, X)}_{\textbf{adversarial losses}} + \underbrace{\lambda \mathcal{L}_{cyc}(G, F),}_{\textbf{cycle consistency loss}} \quad . \tag{2.6}$$

Here, λ controls the influence of the cycle consistency loss.

Regarding the architecture of CycleGAN, the generators are inspired by Johnson et al. [33] and include 3 convolutional layers, several residual blocks, fractionally-strided-convolutions with stride 0.5, and lastly, a convolutional layer that generates the image. Furthermore, they use instance normalization [34]. The discriminators are PatchGANs [35], which classify 70×70 patches of the image into real or fake rather than the whole image and hence, use fewer parameters.

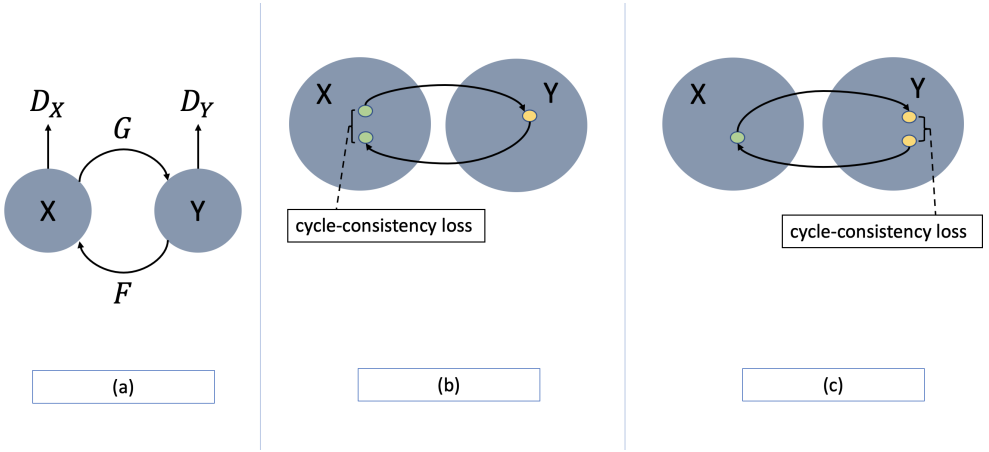

Figure 2.4: (a) Architecture of CycleGAN. (b) Cycle consistency loss controlling $F(G(X)) \approx X$. (c) Cycle consistency loss controlling $G(F(Y)) \approx Y$.

2.1.5 Differentiable Augmentation (DiffAugment)

In order to generate realistic and diverse synthetic data, GANs need a large amount of training data. Collecting such data is expensive, time consuming, and sometimes hard when the objects are rare. However, working with limited data can result in over-fitting and bad performance. To overcome these issues, Zhao et al. [36] introduced DiffAugment, a solution that includes using a data augmentation strategy.

Applying data augmentation to GANs can be tricky. Using data augmentation techniques only on real images will result in learning the augmented data distribution instead of the real distribution. Also, the strategy of augmenting images only during the discriminator updates fails since the generator will not receive any useful updates from the discriminator because it obtains gradients of the fake images without augmentation. To preserve the balance between the generator and discriminator, DiffAugment applies data augmentation to real and fake images in both discriminator and generator updates; see [36, Figure 2.5]. However, the data augmentation has to be differentiable to allow gradient propagation through the augmentation to the generator.

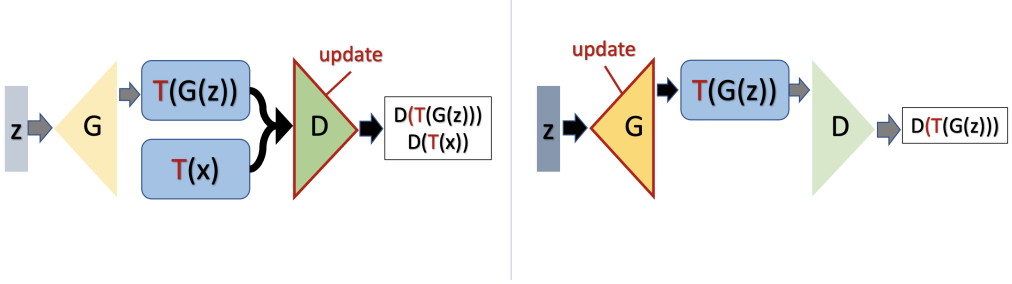

Figure 2.5: DiffAugment strategy applies Augmentation T to fake images $G(z)$ and real images x. To the left, the weights of the discriminator D are updated. To the right, the weights of the generator G are updated.

2.2 Object Detection

Object detection is about answering two questions: What objects are in the image? Where are those objects located? Different systems have been proposed to answer these questions, which can be grouped into One-Stage and Two-Stage Detectors. Two-Stage Detectors, such as the R-CNN family [37], consist of complex pipelines, which use classifiers at various locations in the input image. These systems need many GPUs for training and are hard to optimize since every component of the pipeline must be trained separately. Moreover, to get the final output, the image has to run through several evaluations, making these systems slow.

To make object detection systems faster and achieve real-time performance, Redmon et al. [38] proposed the first One-Stage Detector [39, Figure 2.6] where they reframe object detection as a regression problem. The resulting model, called You Only Look Once (YOLO), is simple, end-to-end trainable, and only needs one evaluation to get the final prediction. Ever since, several improved versions of YOLO have been published, which we will introduce in this section.

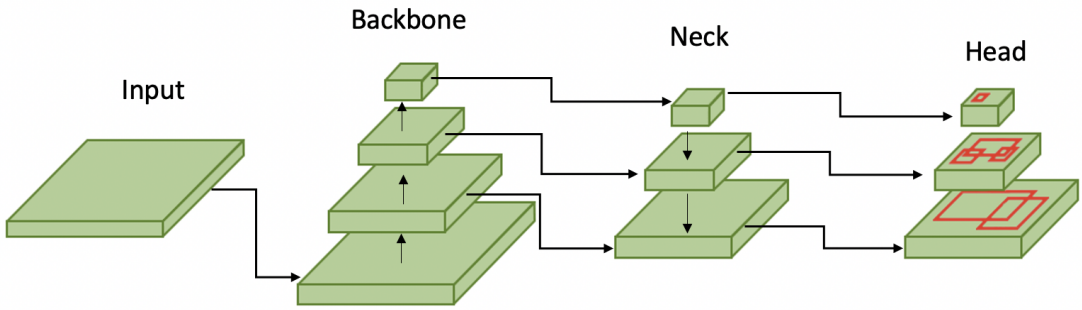

Figure 2.6: A One-Stage Object Detector usually consists of: **Backbone:** A pre-trained neural network that implements the feature extraction task. **Neck:** Included in more recent object detectors; responsible for aggregating features extracted by the backbone. **Head:** The final part of a detector responsible for predicting classes and bounding boxes.

2.2.1 YOLO

YOLO [38] uses one convolutional network, which only needs to look once at the input image to make a prediction.
To make a prediction, the system (illustrated in [38, Figure 2.7]) receives an image as input and splits it into a grid of size $S \times S$. The grid cell, in which the center of an object lies, is responsible for the prediction.

Each grid cell predicts a fixed amount B of bounding boxes. The bounding boxes are defined by four normalized coordinates x, y, w, h, where x and y describe the box's center, while w and h describe the width and height of the box.

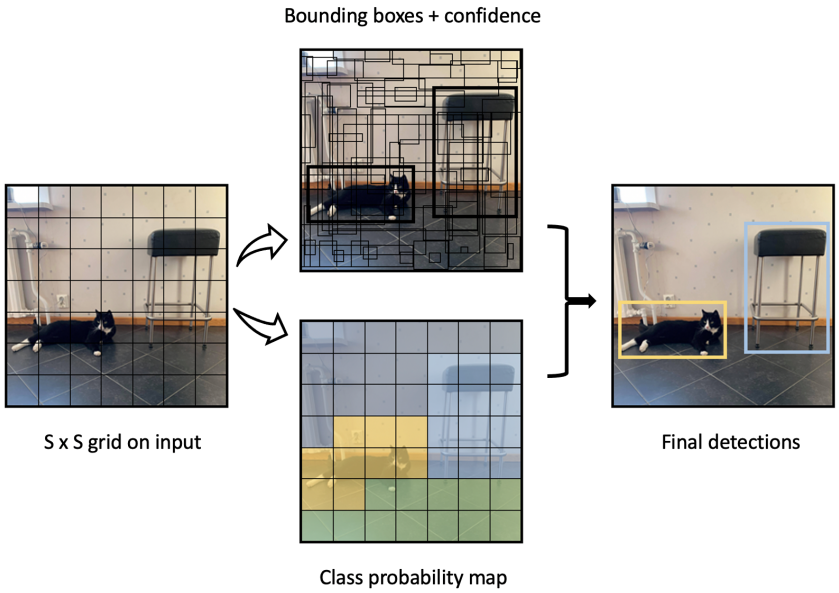

Figure 2.7: The workflow of YOLO illustrated on an example image.

Additionally, each grid cell predicts a confidence score C for each bounding box, which measures the accuracy of the predicted box and the confidence of the model that the box encloses an object. To measure the predicted bounding box's accuracy, the Intersection over Union IoU(g,p) between the ground truth box g and the predicted box p is used, i.e. the area of intersection over the area of union between the two boxes. The resulting confidence score is shown in [38, Equation 2.7].

$$C = P(object) * IoU(g, p). \tag{2.7}$$

Here, $P(object)$ is the probability of an object included in the box, which takes value 0 if no object is present in the box, and value 1 if an object is present. Lastly, each grid cell predicts one set of class probabilities conditioned on the grid cell containing an object, defined in [38, Equation 2.8].

$$p(c_i) = P(class_i \mid object). \tag{2.8}$$

To get class-specific confidence scores for every box at test time, the confidence scores and class probabilities get multiplied, see [38, Equation 2.9].

$$P(class_i) * IoU(g, p) = P(class_i \mid object) * P(object) * IoU(g, p). \tag{2.9}$$

The resulting scores measure the probability of the class being present in that box and the accuracy of the box compared to the ground truth box.

The final output may contain duplicate boxes for a single object. To remove duplicate boxes with lower scores, a method called Non-Maximum-Suppression (NMS) is used. NMS iteratively selects the box with the highest score and calculates the IoU(g,p) with all other remaining boxes. The remaining boxes with IoU(g,p) greater than a given threshold indicate that they might contain the same object and are therefore removed.

The authors implemented the above-described model as a convolutional network with the grid cell and bounding boxes parameters chosen to be $S = 7$ and $B = 2$. The network consists of a backbone with 24 convolutional layers that extract features, followed by a head with two fully connected layers that make the final prediction.

For training, the authors pre-trained the 20 first convolutional layers on the ImageNet competition dataset [1]. After pre-training, the head was added to perform detection and the resolution of the input size was increased from 224×224 to 448×448. The final layer uses a linear activation function and the remaining the leaky rectified linear activation function. To work against overfitting, various data augmentation techniques, including random scaling and randomly adjusting exposure, are used. Additionally, the authors include a dropout layer, which stops layers to co-adapt.

The authors decide to optimize for sum squared error, a loss function used for regression problems that is easy to optimize. The loss compromises three parts: the localization loss term, the confidence score loss term, and the conditional class probability loss term. Since the goal is to maximize average precision, some adjustments have to be made:

1. Sum squared error treats errors in small boxes and large boxes equally, but a small error in a small box should be more punished than a small error in a large box. Therefore, the width and height coordinates in the loss function got replaced with their respective square root to mitigate this problem.
2. Furthermore, sum squared error gives localization error and classification error equal weight, which does not follow the goal of maximizing average precision. To solve this problem, they increase the localization loss term with a factor λ_{coord}.
3. Many grid cells in an image do not contain any object. Hence we train the system more often on detecting background than on actually detecting an object. This can overpower the gradient and can lead to model instability. Therefore, the authors split the confidence loss term into two parts; the sum squared error of boxes with objects present and no objects present. To make the training more stable, the loss from confidence predictions for boxes with no object gets down-weighted with a factor of λ_{noobj}.
4. Finally, the bounding box predictor with the highest current IoU(g,p) with the ground truth box gets assigned for predicting an object. This causes the predictors to specialize and improves the performance of the model.

The resulting loss function is shown in [38, Equation 2.10] and the used symbols are described in Table 2.2.

$$\begin{aligned}
&\lambda_{coord} \sum_{i \in S^2} \sum_{j \in B} \mathbf{1}_{ij}^{obj} [(x_i - \hat{x_i})^2 + (y_i - \hat{y_i})^2 + (\sqrt{w_i} - \sqrt{\hat{w_i}})^2 + (\sqrt{h_i} - \sqrt{\hat{h_i}})^2] \\
&+ \sum_{i \in S^2} \sum_{j \in B} \mathbf{1}_{ij}^{obj} (C_{ij} - \hat{C_{ij}})^2 + \lambda_{noobj} \sum_{i \in S^2} \sum_{j \in B} \mathbf{1}_{ij}^{noobj} (C_{ij} - \hat{C_{ij}})^2 \\
&+ \sum_{i \in S^2} \mathbf{1}_{i}^{obj} \sum_{c \in Classes} (p_i(c) - \hat{p}_i(c))^2
\end{aligned} \tag{2.10}$$

Symbol	Description
$i \in S^2$	Index i refers to cell i in the set of all grid cells S^2.
$j \in B$	Index j refers to bounding box j in the set of all bounding boxes B.
x_i, y_i	The center point of the true box in grid cell i.
w_i, h_i	The width and height of the true box in grid cell i.
$\hat{x_i}, \hat{y_i}$	The center point of the predicted box in cell i.
$\hat{w_i}, \hat{h_i}$	The width and height of the predicted box in cell i.
$\mathbf{1}_{ij}^{obj}$	$\mathbf{1}_{ij}^{obj}$ is 1 if there is an object in cell i and the bounding box j is assigned for making the prediction, and 0 otherwise.
$\mathbf{1}_{ij}^{noobj}$	$\mathbf{1}_{ij}^{noobj}$ is 1 if there is no object in cell i and the bounding box j is assigned for making the prediction, and 0 otherwise.
$\mathbf{1}_{i}^{obj}$	$\mathbf{1}_{i}^{obj}$ is 1 if an object is present in cell i and 0 otherwise.
$p_i(c)$	The true conditional class probability for class c in cell i.
$\hat{p}_i(c)$	The predicted conditional class probability for class c in cell i.
C_{ij}	The true confidence score for box j in cell i.
$\hat{C_{ij}}$	The predicted box's confidence score for box j in cell i.
$c \in Classes$	Class c in the set $Classes$.
λ_{coord}	A factor that increases the localization loss.
λ_{noobj}	A factor to down-weight the confidence score loss when detecting the background.

Table 2.2: List of symbols with corresponding description in the YOLO loss function.

2.2.2 YOLOv2

YOLO suffered from limitations such as localization errors and low recall. To overcome these limitations, Redmon et al. introduced YOLOv2 [40], the second version of YOLO. The goal of YOLOv2 was to enhance the accuracy and make it faster.

To improve the accuracy, several techniques were used:

1. Batch normalization was added to speed up the learning process, and it also removes the need for dropouts.

2. They changed the size of input images from 448×448 to 416×416 to create a feature map of size 13×13, which has a single center cell.

3. The head in YOLO, consisting of fully connected layers, is responsible for the prediction of the bounding boxes, but does not use any prior measurements of shapes. To produce better predictions, the authors decided to use anchor boxes instead of fully connected layers. Anchor boxes allow multiple object predictions per cell by using predefined prior boxes. When moving to anchor boxes, they changed to predict class probabilities for every anchor box instead of only once per grid cell. The best anchor boxes' shapes in YOLOv2 are picked using 5-means clustering to make the learning easier. Hence, the model predicts 5 boxes for each cell. Each prediction consists of 5 coordinates t_x, t_y, t_w, t_h, t_o, which can be used to compute the bounding box coordinates

as shown in [40, Equation 2.11].

$$\begin{aligned} b_x &= \sigma(t_x) + c_x \\ b_y &= \sigma(t_y) + c_y \\ b_w &= p_w e^{t_w} \\ b_h &= p_h e^{t_h} \\ P(object) * IoU(g, p) &= \sigma(t_o). \end{aligned} \tag{2.11}$$

Here, b_x, b_y, b_w, b_h is the bounding box coordinates, c_x, c_y is the distance from the top left corner to the cell, p_w, p_h represents the anchor box's width and height, and σ is a sigmoid activation function (see [40, Figure 2.8]).

4. In order to better detect small objects, a passthrough layer was added to reshape the output from previous layer and then concatenate it with the original output to produce the final output for predictions.

To improve the speed and mitigate the complexity problem, the authors proposed a new classification model called Darknet-19 as the backbone for YOLOv2. The reason for choosing Darknet-19 is having low processing requirement compared to other methods. The number 19 in Darknet-19 refers to the 19 convolutional layers in the network. It mostly uses 3×3 filter for feature extraction followed by 1×1 filters to reduce the feature representation.

YOLOv2 can run on various image sizes, which provides a trade-off between accuracy and speed. However, this version also suffered from a limitation that it did not perform well on small objects.

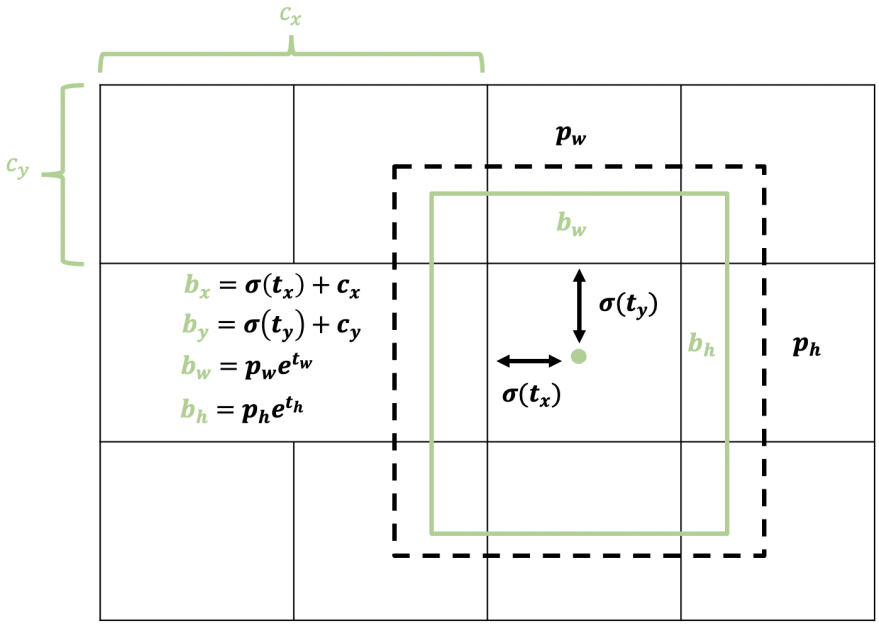

Figure 2.8: Bounding boxes location prediction (green box) with anchor box's dimension (dashed box). c_x, c_y is the distance from the top left corner to the cell, p_w, p_h are the height and width of the anchor box, σ is a sigmoid activation function used for normalization.

2.2.3 YOLOv3

In 2018 Redmon et al. [9] introduced a new version, YOLOv3, that comes with further improvements:

1. The author made changes in the loss function: They changed to predict the object confidence and class predictions with logistic regression instead of the sum squared error used in YOLO; hence the squared error terms got replaced with cross-entropy loss terms. Furthermore, they changed to a multi-label approach for class predictions. Instead of using a softmax function, limiting each box only to have one label, the authors change to use independent logistic classifiers.
2. To help detect smaller objects, YOLOv3 includes a neck that makes predictions at three different scales inspired by Feature Pyramid Networks (FPN) [41]. The idea is to make predictions from the last feature map and also from earlier feature maps in the network. To do so, they upsample an earlier feature map and concatenate it with a feature map from even earlier in the network to combine information from different stages in the network. The combined feature map is then used to make additional predictions. Furthermore, they change to use 9-means clustering to choose the bounding box priors, to have 3 bounding box priors per scale.
3. Lastly, YOLOv3 gets equipped with a new backbone, called Darknet-53. This new backbone is way deeper (53 layers) than the previous Darknet-19 and includes residual blocks, making it a powerful but still efficient network.

2.2.4 YOLOv4

The latest version of YOLO; YOLOv4 [39] is an object detector addressing the issues of previous object detectors by enhancing the accuracy of a Convolutional Neural Network (CNN) that can operate using a conventional GPU. YOLOv4 achieved an Average Precision (AP_{50}) value of 65.7 percent on the MS COCO dataset [42], and a speed of about 65 frames per second (FPS) on the Tesla V100 GPU. It exceeds the existing models significantly in both performance and speed and can be trained smoothly and operate fast in production systems.

The main goal of designing a new version of YOLO is to allow anyone who uses a conventional GPU to train and test on custom data. This detector needs to combine speed and accuracy to perform real-time object detection with high quality. Many features can be used to enhance a CNN, and since the detector has to contain features that contribute to both speed and accuracy, several combinations of features were tested by the authors to obtain the best set.

To achieve the optimal balance, the authors focused on choosing the architecture of YOLOv4 carefully. Having a good accuracy means we need to process images with good resolution, maintain the spatial information along with training, learn useful features while having a suitable number of layers in the CNN. Fast means we need ways to speed up the training process without losing essential features from the input image. The authors did many experiments to choose a suitable architecture

that can achieve the main goal. The final selection included an interesting addition of features that overcame previous limitations and produced a state-of-art detector:

Backbone: YOLOv4 uses CSPDarknet53: The Cross-Stage-Partial (CSP) [43] connections are combined with the Darknet53 as the backbone for feature extraction. The reason behind using CSP connections is to reduce the computations and improve the learning ability of Darknet53. CSP achieves that by splitting the input feature map into two parts and then combine it later using a cross-stage strategy.

Neck: YOLOv4 uses Path Aggregation Network (PANet) [44] for feature aggregation. PANet helps maintain spatial information exactly, allowing additional connection between the lower and upper layers and the ability to access features from all layers. As the image goes through various layers in the neural network, the complexity increases, and the pixel location is lost, and thus it becomes difficult to identify these features at higher levels accurately. The authors modified PANet by concatenating the neighboring layers instead of adding them, which improves the accuracy of predictions. The addition of PANet helps in accessing the features from lower layers faster. The previous YOLO version used an FPN aggregator that performs a top-down approach, but this process becomes very slow, especially when we have hundreds of layers. To improve the speed, PANet performs a bottom-up approach so that we can access features from lower layers quickly.

Head: The head is the final part that makes the prediction. YOLOv4 uses YOLOv3's (anchor-based) head.

YOLOv4 is a detector that proved to be faster and more accurate than previous versions due to a careful selection of techniques, which significantly improved the detector's performance and the classifier. These techniques can be grouped into two groups: Bag of freebies and Bag of specials.

2.2.4.1 Bag of freebies and Bag of specials

Methods that can improve the detector accuracy without causing any latency at inference time are called *bag of freebies* (BoF), while methods that can drastically improve the accuracy but increase the inference time are *bag of specials* (BoS). The authors of YOLOv4 tested various combinations of these methods. Below we list the features that the authors included in their model.

Bag of Freebies (BoF)[2]:

1. **CutMix:** This technique combines parts from images together into an augmented image. The model will learn how to make predictions based on a larger amount of features [45].
2. **Mosaic data augmentation:** This data augmentation technique combines 4 training images into one image. It helps to identify objects at a smaller scale rather than a normal scale.

[2]BoF 1, 2, 3 are used during the pre-training of the backbone, while 2, 5, 6, 7, 8 are used during the detector training.

3. **Class label smoothing:** This technique reduces overfitting by encoding values with some uncertainty. For example, instead of using 1.0 for the most likely class, one can use 0.9 [46].
4. **Complete IoU(g,p) loss (CIoU):** This loss treats the bounding-box coordinates: center point, height, and width as one unit, and thus the prediction is more accurate [47].
5. **Eliminate grid sensitivity:** In YOLOv3 [9] the equations $b_x = \sigma(t_x) + c_x$ and $b_y = \sigma(t_y) + c_y$ are used to predict the center coordinates, where c_x, c_y are whole numbers giving the distance to the left corner. For b_x to be predicted as c_x or $c_x + 1$, t_x has to be a very large negative or positive number respectively. To address the problem of having spots in the grid that are hard to predict, the authors of YOLOv4 multiply the sigmoid output with a factor greater than 1.
6. **Using multiple anchors for single ground truth:** YOLOv3 only assigned one anchor box for each given object. In YOLOv4, the authors changed to use multiple anchors for each ground truth object. An anchor box is responsible for the detection of a given object if the IoU(g,p) between the two boxes is greater than a given threshold.
7. **Random training shapes:** Removing fully connected layers from YOLO made it possible to train with various image sizes. Multi-scale training makes the model more robust and enhances generalization.
8. **Batch Normalization (BN):** Batch Normalization (BN) [48] is the common technique of normalizing the mean and variance of the input to each layer to work against the internal covariate shift.

Bag of Specials (BoS)[3]:

1. **Mish activation:** YOLOv4 uses the Mish activation function introduced by D. Misra [49]. Mish outperformed other activation functions like ReLU in different computer vision tasks. It has a low cost and includes various properties such as smooth and non-monotonic nature, unbounded above, which avoids saturation and increases accuracy, and bounded below, which helps achieve strong regularization effects and reduces overfitting.
2. **Spatial Attention Module block (SAM-block):** Spatial Attention Module (SAM) [50] is an attention module that encodes the important features in an image. The feature map goes through two transformations using max and average pooling layers. Then, the concatenated features are fed into a convolutional layer. Finally, a sigmoid function is applied. In YOLOv4, a modified version of SAM is used called SAM-block, which does not include pooling layers.
3. **Modified Spatial Pyramid Pooling Layer (SPP-block):** To enhance the receptive field of the backbone, YOLOv4 uses an adjusted version of the SPP module introduced by He et al. [51]. The adjusted SPP-block [52] performs max-pooling layers to generate different representations of feature maps and then concatenates them again.

[3]BoS 1 is used during the backbone's pre-training, while 1, 2, 3 are used during the detector training.

2.3 Evaluation Metrics

We evaluate in two stages of our project: firstly, we evaluate the generated outputs of the GANs, and secondly, we evaluate the performance of the Object Detection system.

For the evaluation of the GANs, we will do human evaluation, while the evaluation of the Object Detection system will include metrics such as mean Average Precision (mAP), Loss Function, Confidence Score, Precision, and Recall. We will use the metrics during training to monitor the system's performance, which allows us to review and adjust accordingly. Furthermore, we will use the already implemented model in KSO as a baseline model and compare the performance to our model.

2.3.1 Mean Average Precision (mAP)

The mAP is the standard metric used to evaluate object detection systems. It is a metric that combines Precision and Recall, which are calculated at a specific IoU(g,p) threshold, which is the area of overlap of the true and predicted boxes divided by the union of the two boxes. The mAP at this certain threshold is defined as the mean of the Average Precision (AP) over all classes, where the AP is the mean Precision at eleven recall levels [53]. Depending on different detection challenges, the mAP is defined differently. A commonly used threshold is 0.5, which is the threshold we will consider, denoted by mAP@0.5.

2.3.2 Loss Function

The Loss Function helps us to adjust weights to decrease the cost. When there is a wrong prediction, the Loss Function gives us direction on where to move. The loss in YOLOv4 consists of three parts, the loss for the bounding box predictions, the loss for the class predictions, and the loss for the objectness score, where the last two parts are the same as in YOLOv3. Traditionally the L2 standard loss function is used to make bounding box predictions, but recently researchers introduced the IoU loss [54], which treats the bounding box's coordinates as a unit. Thus IoU loss provides a more accurate prediction. YOLOv4 uses the Complete IoU Loss (CIoU) [47], which reduces the distance between the central points of the true and predicted box and increases their overlap.

2.3.3 Confidence Score

YOLO predicts an confidence score reflecting the accuracy of the predicted box and the confidence of the model that the box encloses an object. For each box, it also predicts a set of class scores. At test time, YOLO outputs a class specific confidence score calculated by multiplying the confidence score and class probabilities. The confidence score measures the accuracy of the predicted box and the probability of a certain class being present in the box.

2.3.4 Precision and Recall

A True Positive (TP) denotes that the algorithm correctly detected an object. A False Positive (FP) indicates that the detected object was classified into the wrong category, for example, classifying a deepwater coral as a sponge. A False Negative (FN) indicates that the algorithm failed to detect a present object in an image, for example, when the algorithm fails to detect a deepwater coral [55].

Precision and Recall are two ratios that are defined in [56, Equation 2.13] and [56, Equation 2.13].

$$Precision = \frac{TP}{TP + FP} \tag{2.12}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.13}$$

Here, Precision can be seen as a measure of quality and Recall as a measure of quantity. Precision is the fraction of correctly detected objects among all detected objects, while Recall presents the fraction of detected objects among all ground-truth boxes [56].

3
Methods

This study aims to build a robust system that can achieve more accurate detection for deepwater corals and generalize to unseen data in different environmental conditions. To obtain a representative training dataset, we tested several GANs to enhance (Section 3.2) and increase (Section 3.3) our limited data (Section 3.1). After pre-processing our data, we trained a YOLOv4 object detector and tested it on unseen data (Section 3.4).

3.1 Data

Our labeled training data consists of 409 deepwater corals (Desmophyllum pertusum) images of size 720×576. The images were taken by ROVs in the Kosterhavet National Park and annotated using the citizen science module from KSO. As shown in Figure 3.1, most of the images contain white deepwater corals on dark background. The image quality varies due to lack of light, marine snow, etc.

Figure 3.1: Deepwater coral examples from our training data. Good quality to the left, bad quality to the right.

3.2 Image Enhancement

The first step in our pipeline was to enhance the data so that the detector could recognize the deepwater corals more easily. To improve the quality of the data, we trained a CycleGAN, which is used to translate between two domains. We set domain X to represent our poor quality images and domain Y to represent a set of good quality images. Then, we trained the network to learn the mapping $G : X \rightarrow Y$, which aims to enhance our images.

To prevent the CycleGAN from over-fitting to our training data and performing poorly on testing data, we used different data than our 409 deepwater coral frames

for training. Since we did not have additional deepwater coral frames, we used frames of deeplet sea anemones, *Bolocera tuediae (Johnston, 1832)*, taken by ROVs in the Kosterhavet National Park as domain X. For domain Y, the images should be similar to domain X in terms of the objects present in the images to successfully learn a mapping that only enhances the images. However, we lacked good quality images of deeplet sea anemones taken in the Kosterhavet, and therefore decided to use hand-picked images from ImageNet [1] including the classes *sea anemone* and *coral*. We added corals to domain Y since the deeplet sea anemone dataset for domain X also included corals in some frames. Besides, we received some high-quality underwater images from MMT Sweden AB for our research, which we included in domain Y. The datasets' resulting sizes for domain X and Y were 409 and 304, respectively. Examples for both domains are illustrated in Figure 3.2.

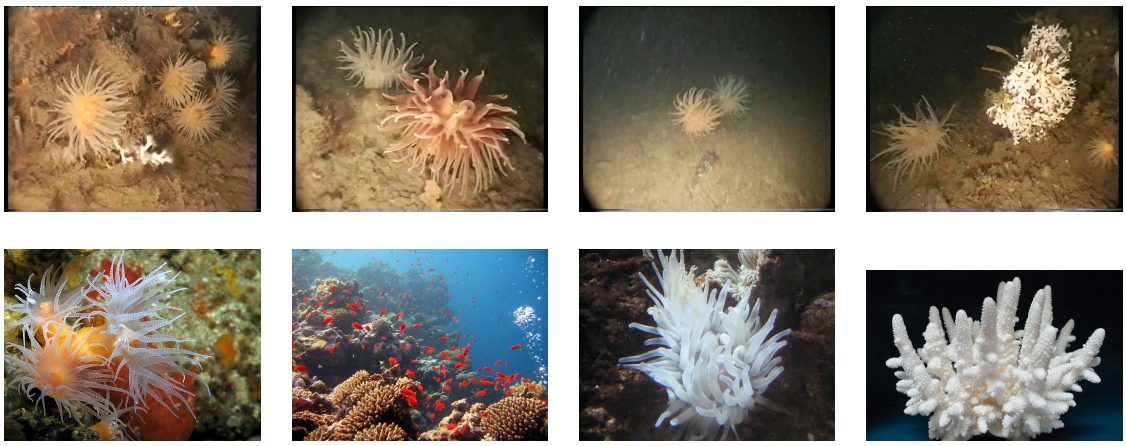

Figure 3.2: Domain X images (first row). Domain Y images (second row), Source: [1].

3.2.1 Training

We used the PyTorch implementation from the CycleGAN authors [32]. To stabilize the training, we followed the authors and used the least-square GANs [57] objective and updated the discriminator based on the last 50 enhanced images instead of only using the latest ones.

We resized our images to 320×320. We chose this size to fulfill the requirements of both CycleGAN, which requires image sizes divisible by 4, and YOLOv4, which requires sizes divisible by 32. Moreover, we chose 320×320 as this is the largest possible size for the images in domain Y. In the generator's architecture, we used 9 residual blocks following the author's implementation for image sizes larger than 256. We trained the networks and evaluated the results for different epochs to decide when to stop the training. We stopped training when the enhanced images did not change significantly in further epochs. For all other hyper-parameters, we chose the default values used by CycleGAN: We set the regularization term λ in the loss to 10. We used an Adam optimizer with $\beta_1 = 0.5$ and $\beta_2 = 0.999$. We used an initial learning rate of 0.0002 and linearly decay it to 0 after 100 epochs.

3.2.2 Evaluation

Performance evaluation of GANs is still an open research question, and there is no single standard metric, which applies to all tasks and datasets [58]. One common method is to use human annotators to judge the realness and quality of generated images. Therefore, we decided to evaluate the enhanced images depending on our predetermined criteria. We checked if the enhanced images have:

1. less noise/marine snow,
2. less blur and more sharp edges,
3. reduced color distortion (i.e. the deepwater corals appear white, the image has natural colors, etc.).

3.3 Image Augmentation

The second step in our pipeline was to create a large and diverse training dataset that could be used to train the object detection system. We did this by using GANs to generate synthetic data.

First, we decided to experiment with two models, DCGAN [23] and WGAN-GP [26]. This selection is based on a tradeoff between choosing GANs with simple architectures and low computational cost but that are also stable at training and align with our goals for this task. The generator and discriminator architectures used in both GAN models are designed to output images of size 64×64. Images of this size might be too small to train the object detector successfully. However, we first focused on a smaller size because training GANs is very difficult and gets even harder with increased size due to increased training instability. [29].

After experimenting with the above mentioned models, we decided to use a more advanced model in order to output images of higher resolution; we chose to use StyleGAN2 [31].

Finally, since GANs require a large amount of training data, and we were constrained with limited data, we used the data augmentation strategy DiffAugment [36] for all models.

3.3.1 Training

3.3.1.1 DCGAN and WGAN-GP

For DCGAN, we used the PyTorch implementation by Nathan Inkawhich [59], which follows the DCGAN authors' implementation. The model was trained using mostly the same parameters as suggested by the authors: We used an Adam optimizer with momentum parameters $\beta_1 = 0.5$ and $\beta_2 = 0.999$ and learning rate 0.0002. The weights were initialized using a normal distribution with mean 0 and standard deviation 0.02. For the Leaky-ReLU Layers in the discriminator, the slope was set to 0.2.

For WGAN-GP, we took the PyTorch implementation from [60] and made adjustments to meet the paper's suggestions. We used the same architecture for the generator as in the DCGAN implementation. However, the DCGAN discriminator was adjusted by dropping the last layer to form the critic network. Moreover, we used instance normalization in the critic instead of batch normalization to prevent the conflict with the gradient penalty, as suggested by the authors. After experimenting with some hyper-parameters, we again followed the default hyper-parameters for training, since it worked best: We used an Adam optimizer with momentum parameters $\beta_1 = 0$ and $\beta_2 = 0.9$ and learning rate 0.0001. We did 5 iterations for the critic before updating the generator and used a penalty coefficient set to 10. The weights were initialized the same way as in the DCGAN training.

After experimenting with different batch sizes for both models, we found that a batch size of 32 worked best in our case. To decide when to stop training, we generated images from a fixed noise every 20 epochs. We stopped training when the images did not improve.

3.3.1.2 StyleGAN2

For StyleGAN2, we used the PyTorch implementation by Zhao et al. [36]. StyleGAN2 can be trained to generate images of size 4×4 up to 1024×1024, where the sizes double each time. However, since our training images are of size 720×576, we trained the network to generate images of size 512×512, the highest possible resolution for us. The model was trained with the implemented default hyper-parameters: We used an Adam optimizer with momentum parameters $\beta_1 = 0$, $\beta_2 = 0.99$ and learning rate 0.002 for all weights, except for the mapping network, which used 100 times lower learning rate. Furthermore, the implementation includes an equalized learning rate approach[1] [29]. For the objective function, StyleGAN2 uses the improved loss from the original GAN paper [20] together with R_1 regularization[2] [61] with regularization parameter $\gamma = 10$. As activation function, leaky ReLU was used in both the discriminator and generator with a slope set to $\alpha = 0.2$.

We trained the network with a batch size of 8 for a total of $500k$ image iterations (about 1222 epochs). We chose this training length following the *Low-Shot Generation Experiment* [36] on the AnimalFace dataset [62] with similar size. During training, we generated images every $40k$ iteration. For the final model, we picked the weights for which the quality of generated images stopped improving. When generating images, the exponential moving average of the generator weights [29] with decay 0.999 was used to reduce substantial weight variations between training iterations.

[1] In the equalized learning rate approach, all weights are initialized from $\mathcal{N} \sim (0, 1)$ and scaled per-layer using a normalization constant during training. This approach is useful since the weights then have a similar scale during training, and hence, the learning speed is the same for all weights.

[2] R_1 regularization stabilizes the training process by penalizing the discriminator for deviating from the optimum: $R_1 = \frac{\gamma}{2}\mathbb{E}_{x\sim\mathbb{P}_r}[\|\nabla D(x)\|^2]$

3.3.1.3 DiffAugment

For DiffAugment, we used the PyTorch implementation provided by the paper [36]. Their implementation includes three simple augmentation techniques: *Color*, *Translation* and *Cutout*. *Color* includes adjusting brightness, saturation, and contrast. *Translation* involves resizing the image and padding the remaining pixels with zeros to display the objects in different positions. *Cutout* cuts out a random square of the image and pads it with zeros. We used all three transformations as recommended by the authors when training with limited data.

When experimenting with DCGAN and WGAN-GP, we decided to run both models with and without DiffAugment to test the advantage of using such an augmentation technique. For StyleGAN2, we only trained the model with DiffAugment based on the results from the above experiment. Hence, we trained the following five models:

1. DCGAN
2. DCGAN + DiffAugment
3. WGAN-GP
4. WGAN-GP + DiffAugment
5. StyleGAN2 + DiffAugment

3.3.2 Evaluation

Different approaches can be utilized to evaluate the quality of the GANs' generated images. Besides human evaluation, metrics like the Frechet Inception Distance (FID) [63] have been introduced. FID is an evaluation metric comparing the distribution of the generated images with the real images by computing the Frechet distance between the two distributions. However, datasets of at least 10000 images are recommended to get an insightful FID score; otherwise, the score is underestimated. Since our training datasets only consist of 409 images, we decided to use human evaluation to judge the generated images' visual quality. We used our own common sense to determine if generated images are realistic, diverse, and represent the training data.

3.4 Object Detection

3.4.1 Data Preparation

To prepare the data for training, we generated 3000 synthetic images with StyleGAN2, the resulting best model from the data augmentation evaluation, see Section 4.2.2. Using labelImg [64], we annotated the images and filtered out unrealistic images and images not including any coral, so we ended up with 2266 synthetic images. We combined the 2266 synthetic images with the 409 real images, which resulted in a final dataset of 2675 images. We split the data into two parts: 90% for training (2407) and 10% for validation (268). We performed a stratified split to preserve the proportion of real and synthetic data in both the training and validation dataset.

3.4.2 Training

To train the YOLOv4 model on our dataset, we used the darknet [65] framework provided by the authors [39] [9]. We applied transfer learning by using the pre-trained weights for the convolutional layers of the model trained on MS COCO [42].

We trained two networks; the first network, named YOLOv4-baseline, consists of a CSPDarknet53 backbone with Mish activation, PANet neck with leaky ReLU activation, and the YOLOv3 head. It uses all Bag of freebies and Bag of specials for the detector training, as mentioned in Section 2.2.4.1, except the attention module SAM. The second model, named YOLOv4-SAM-Mish, additionally integrates SAM and uses the Mish activation on the PANet neck.

We followed the instructions of the authors and used the default configurations for both networks training: We set the network width and height to 512×512, so every image was resized to that size during training and detection. The optimizer used in YOLOv4 is Stochastic Gradient Descent (SGD) with momentum hyperparameter 0.949 and weight decay 0.0005. The initial learning rate in YOLOv4-baseline and YOLOv4-SAM-Mish was set to $lr = 0.001$ and $lr = 0.0013$, respectively. In the first 1000 iterations, called the burn-in, the learning rate was changed to $lr * (\frac{iteration}{1000})^4$. After the burn-in, the initial learning rate was used. At iteration 4800 and 5400, the learning rate was multiplied by a factor of 0.1. To eliminate grid sensitivity, the scaling factor of the sigmoid outputs in each of the three prediction heads was set to $1.2, 1.1$, and 1.05, respectively. The IoU(p,g) threshold, which determines if an anchor box is used for a ground truth object, was set to 0.213. During training, the images were randomly cropped and resized with changing aspect ratio from $1 - 2 * jitter$ to $1 + 2 * jitter$, where $jitter$ was set to 0.3. To train YOLOv4 with different resolutions, the network was resized randomly after each 10 batches by a factor ranging from 1/1.4 to 1.4.

Several data augmentation techniques were used during training, including randomly adjusting Saturation, which indicates the intensity of a color, Hue, which determines the actual color (red, blue, yellow, etc.), and Exposure, which refers to the amount of brightness in an image. We trained using the default values: The Saturation parameter was set to 1.5, meaning that the Saturation gets multiplied by a factor ranging from $1/1.5$ to 1×1.5, the Hue parameter was set to 0.1 meaning that color gets multiplied by a factor ranging from -0.1 to 0.1, and the Exposure parameter was set to 1.5 meaning that brightness gets multiplied by a factor ranging from $1/1.5$ to 1×1.5. Furthermore, Mosaic, which combines 4 training images into one image, Mixup, which generates a new image by combining two random images, and Blur, which randomly blurs the background 50% of the time, were used during training.

We trained the networks with a batch size of 64 for a total of 6000 batch iterations. We chose a mini-batch size[3] of 2, since larger mini-batch sizes were not possible with the RAM available on the school server. After the burn-in, the mean Average

[3]The mini-batch size determines how many images are processed at once. The weights will be updated after one whole batch sample.

Precision at threshold 0.5 (mAP@0.5) was calculated for each 4^{th} epoch on the validation set, which we used together with the loss to decide when to stop training.

3.4.3 Evaluation

To evaluate the two models - YOLOv4-baseline and YOLOv4-SAM-Mish - we compared the mAP@0.5 calculated on the validation dataset for the final chosen weights. Furthermore, we compared True Positives (TP), False Positives (FP), and False Negatives (FN) at different confidence thresholds. Taking the mentioned validation metrics into account, we decided on a final model.

3.4.4 Testing

To answer our research questions, we conducted three experiments and compared the results of our model with the model implemented in the KSO system [7]:

Testing on data with the same environmental settings

We conducted the first experiment to test if our model can overcome the overfitting issue on data from the Kosterhavet National Park with the same environmental settings. To do so, we got a 30-minutes movie, which holds similar characteristics and environmental settings as the training data. The movie mainly contains white deepwater corals on black background with varying image quality due to marine snow and lack of light.

To conduct the experiment, we split the movie into frames and labeled the frames using labelImg [64]. The labeled frames contained 542 true boxes. To compare our model with the KSO baseline model, we computed the mAP@0.5, TP, FP, FN, Precision, and Recall for both models. To further evaluate the models' performance and understand when the models had problems, we printed out the images with predicted boxes and compared them.

Testing on data with different environmental settings

The second experiment was to test if our model can generalize to data inside the Kosterhavet National Park but with different environmental settings. For this purpose, we obtained three 10-minutes movies, which hold different characteristics and technical environmental settings. The movies were recorded using a more advanced camera where the footage seems brighter, has better quality, and the background and seafloor are more visible than in the training data.

We split the movies into frames and labeled them using labelImg [64]. The labeled frames contained 56 true boxes. To compare the results of our model with the results from the KSO baseline model, we computed the mAP@0.5, TP, FP, FN, Precision, and Recall for both models. Moreover, we printed out images with predicted boxes to compare them and highlight the detection problems.

Testing on data from outside the Kosterhavet National Park

The last experiment was to investigate if our model can generalize to different environments outside the Kosterhavet National Park. Unfortunately, we did not get access to underwater footage from other areas, so we decided to use the coral images from MMT Sweden AB for this purpose. These 17 high-resolution images were taken outside the Kosterhavet National Park with a drop camera, which makes the environment differ not only in terms of location but also in terms of the camera settings (e.g., different angle and lighting, better quality).

To run our experiment, we first labeled the images with labelImg [64], which resulted in 100 boxes containing corals. After labeling, we computed the mAP@0.5, TP, FP, FN, Precision, and Recall for our model and the KSO-model and compared the results. Finally, we also printed out the images containing the predicted boxes to further understand the models' behavior.

4

Results and Discussion

4.1 Image Enhancement

4.1.1 Training

We trained CycleGAN for a total of 300 epochs. Figure 4.1 and Figure 4.2 show results of two deepwater coral images for different epochs. We can see that the model was still learning up to epoch 250 and, for example, the colors changed. Some images, like Figure 4.1, did not change much after epoch 250, while other images, like Figure 4.2, still changed from epoch 250 to 280. Therefore, we decided to take the weights of epoch 280 for our final model.

Figure 4.1: Example 1: Real image and corresponding results for epoch 100, 200, 250, 280 and 300.

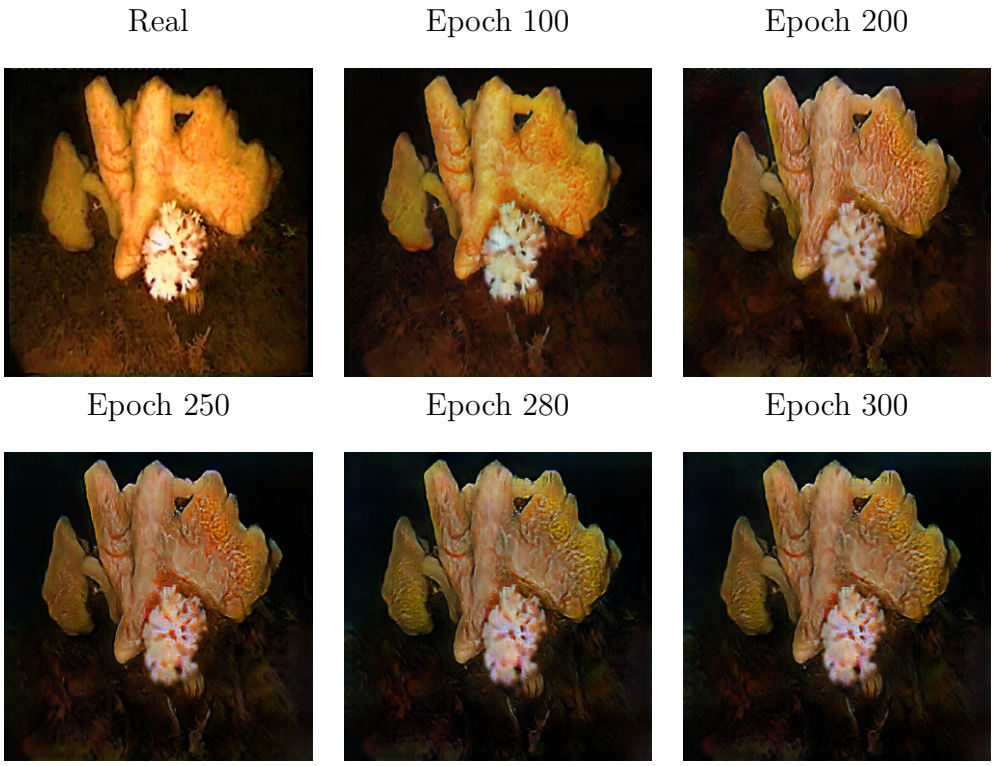

Figure 4.2: Example 2: Real image and corresponding results for epoch 100, 200, 250, 280 and 300.

4.1.2 Evaluation

We enhanced our 409 deepwater coral images using the weights from epoch 280. Then, we went manually through the real and enhanced images and checked if the aforementioned criteria matched. The following observations were made:

For some images, the color distortion got reduced; for example, the deepwater corals appeared more in their actual color like in Figure 4.3. Moreover, in some images, the background contained less marine snow compared to the real images, as the second-row example in Figure 4.3. Some deepwater corals also had sharper edges, and the overall image was less blurry, see Figure 4.1. Even though some images' quality improved through the enhancement process, the results were not perfect, and there is still much room for improvement.

Unfortunately, for many images, the transformation did quite the opposite of enhancing. Figure 4.4 shows two examples where the images got very colorful, unrealistic and the deepwater coral disappeared. We think a reason for this might be that domain X and domain Y differ more than just in terms of quality since the images do not come from the same area and have different characteristics. However, we lack good quality images from the Kosterhavet National Park, so choosing images from ImageNet was the best option available to us.

To successfully build an enhancement system using CycleGAN, we think that a set of good quality images from the Kosterhavet region is needed. Unfortunately, we do

not have that kind of images available to us and acquiring such data would take time. Therefore, we will focus on increasing the unprocessed images in the augmentation step and leave the research about building an enhancement network for the KSO pipeline for future work.

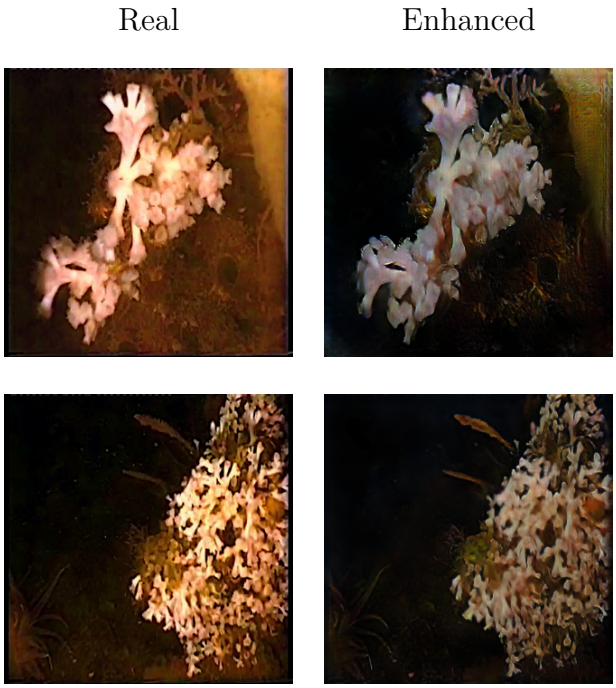

Figure 4.3: Examples where the enhancement succeeded.

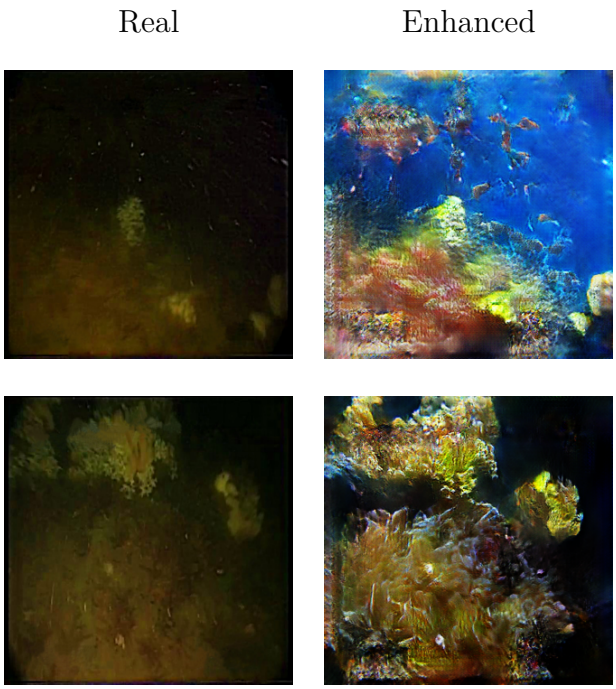

Figure 4.4: Examples where the enhancement failed.

4.2 Image Augmentation

4.2.1 Training

4.2.1.1 DCGAN and WGAN-GP

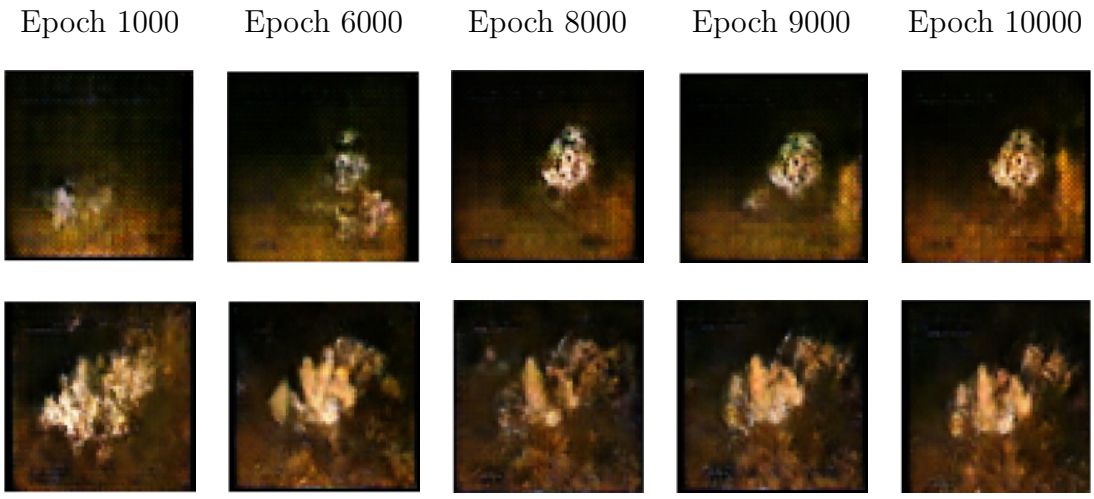

Figure 4.5: Example output for different epochs: WGAN-GP + DiffAugment (first row), DCGAN + DiffAugment (second row). Both models were trained on unprocessed data.

Figure 4.5 shows example output for different epochs for the models trained with DiffAugment on unprocessed data. For WGAN-GP we believe that epoch 9000 gives the best result. For DCGAN we chose the weights from epoch 6000 for our final model. We think that these epochs reflect the most realistic images, and the generator does not improve significantly afterward.

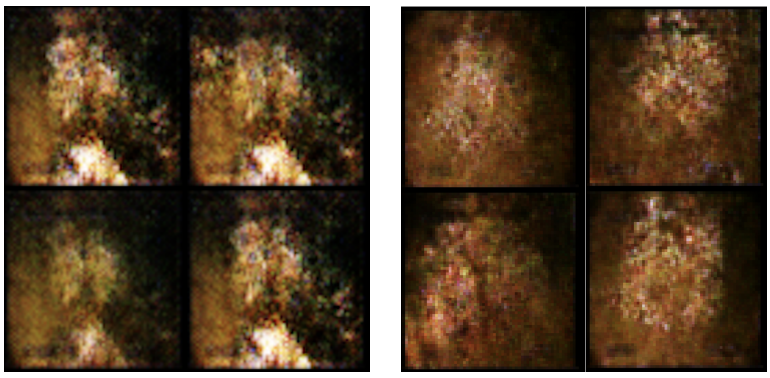

Figure 4.6: Examples of generated images of DCGAN (left) and WGAN-GP (right) both without DiffAugment. Different noise gets mapped to the same image (mode collapse).

The models trained without DiffAugment suffered from mode collapse, meaning that the generated images are not very diverse. Figure 4.6 shows example output for DCGAN and WGAN-GP models trained for 500 and 2000 epochs, respectively. We observed that DCGAN got into mode collapse way earlier than WGAN-GP, which is expected since the WGAN-GP objective is more robust against this issue. However, these results show that we rely on a technique like DiffAugment to train GANs on limited data. We, therefore, focused only on evaluating the results from models trained with DiffAugment.

4.2.1.2 StyleGAN2

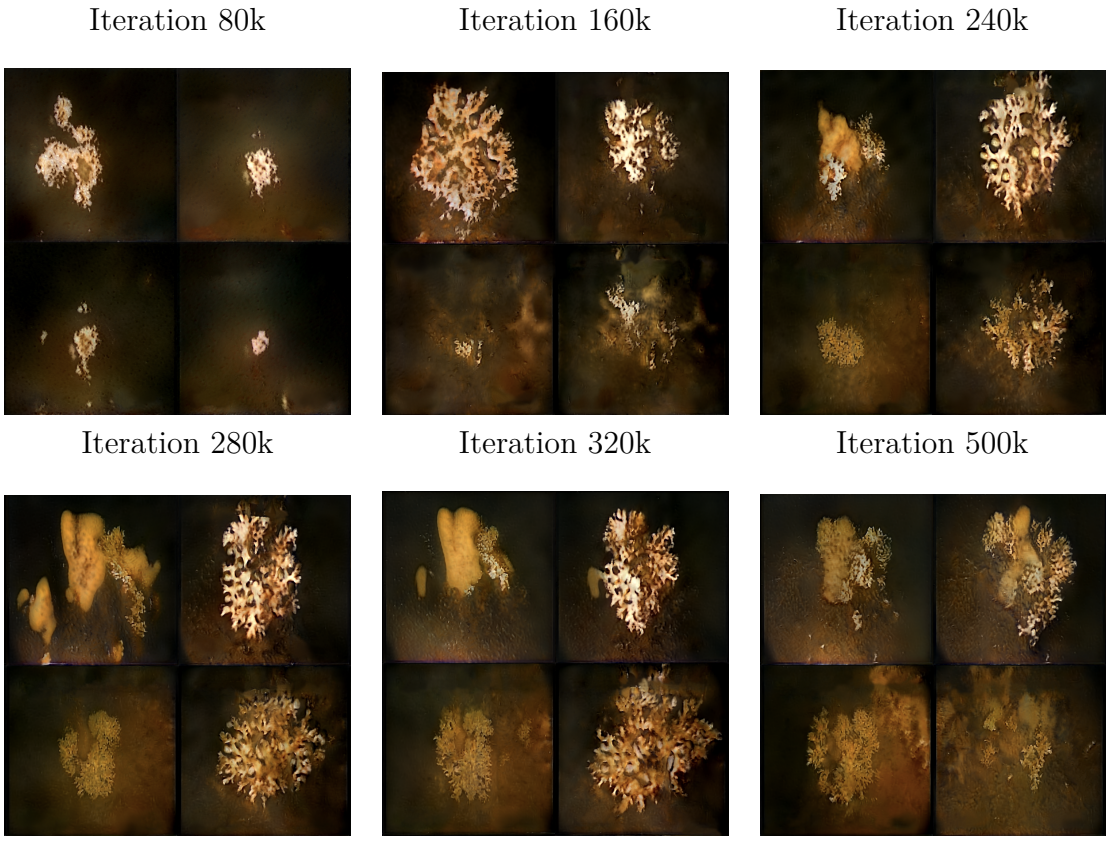

Figure 4.7: Examples of generated images of StyleGAN2 + DiffAugment after $80k$, $160k$, $240k$, $280k$, $320k$ and $500k$ image iterations.

We generated a grid containing 120 images during the training process after every $40k$ image iterations, Figure 4.7 shows example output after $80k$, $160k$, $240k$, $280k$, $320k$ and $500k$ image iterations. In iteration $80k$, we can see mostly white dots on the black and blurry background, while in iteration $160k$, the white dots evolved to form some of the corals' characteristics. After that, in iteration $240k$, we believe that the corals' characteristics became clearer, and the images started to contain stones as well. However, the corals contained some odd dots, see top right corner image of iteration $240k$ in Figure 4.7. In iteration $280k$, we think that the images further improved: the corals, stones, and the background appeared more realistic.

For later iterations like $320k$ and $500k$, we noticed that the images did not improve much compared to iteration $280k$. In contrast, in later iterations, we believe that the quality decreased, and the model created images containing blue artifacts, like demonstrated in Figure 4.8. Due to all mentioned observations, we decided to use the weights of iteration $280k$ for our final model.

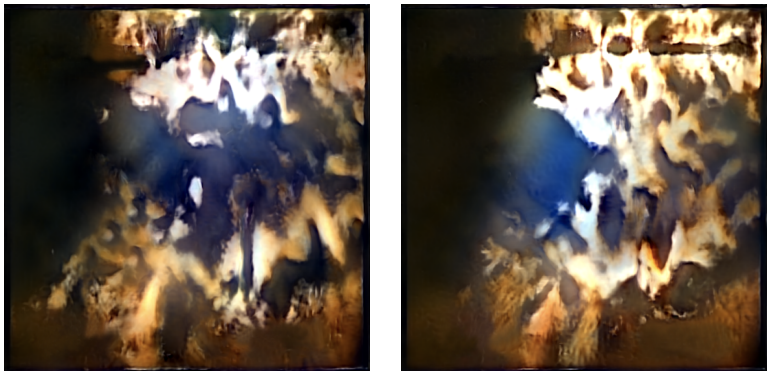

Figure 4.8: Example image with artifacts for StyleGAN2 + DiffAugment after $400k$ (left) and $500k$ (right) image iterations.

4.2.2 Evaluation

We compared generated images with training data for DCGAN, WGAN-GP, and StyleGAN2 trained with DiffAugment and found that all generators learned features of the training data:

Looking at generated images from DCGAN, WGAN-GP, and StyleGAN2 (Figure 4.9), we can see that all models learned the characteristics of deepwater corals, the black background, the brown sea ground, and other objects such as stones, grass, etc. Furthermore, the images vary, which means that different noise vectors are not mapped to the same image. Some generated images are very blurry and do not contain any objects, but these kinds of images are also included in the training data, so it makes sense that the model generates that kind of data. When comparing the results from DCGAN and WGAN-GP, we do not see a significant difference in quality nor variety. However, the results from StyleGAN2 show better quality and appear more realistic than the results from both DCGAN and WGAN-GP.

To conclude, all models trained with DiffAugment learned features of the underlying training data distribution. These results so far suggest that GANs can be used to increase the training data synthetically. However, DCGAN and WGAN-GP can only generate images of size 64×64. In order to train the object detector successfully, we need images with a higher resolution. Since StyleGAN2 produced better quality images and can generate images of size 512×512, we decided to increase our training data using StyleGAN2.

Figure 4.9: Samples of training images (left) and generated images (right) for DCGAN (first row), WGAN-GP (second row) and StyleGAN2 (last row).

4.3 Object Detection

4.3.1 Training

We trained YOLOV4-baseline and YOLOv4-SAM-Mish for 6000 batch iterations and visualized their respective average loss and mAP@0.5 curves, shown in Figure 4.10 and 4.11 respectively. The average loss was computed on the training set, while the mAP@0.5 was computed on the validation set. The average loss decreased considerably until iteration 1300 for both models and then dropped gradually to reach a final average loss of 0.3776 for YOLOV4-baseline and 0.3712 for YOLOv4-SAM-Mish. The mAP@0.5 clearly increased until iteration 2000 with a mAP@0.5 of $\approx$ 95% for YOLOV4-baseline and mAP $\approx$ 96% for YOLOv4-SAM-Mish. For YOLOV4-baseline, the mAP@0.5 remained rather stable and reached $\approx$ 95% at iteration 6000. For YOLOv4-SAM-Mish, the mAP@0.5 slightly dropped to $\approx$ 93% at iteration 4400 and increased again to reach $\approx$ 95% at iteration 6000.

The decreasing average losses indicate that both models learned features of the training data, and the mAP@0.5 curves denote that the models did not overfit to the training data. Although the YOLOv4-SAM-Mish model has a slightly better mAP@0.5 at batch iteration 2000, following the YOLO authors' recommendations for training, we think that 2000 batch iterations are not enough; hence, we decided to take the last weights at batch iteration 6000 for both models.

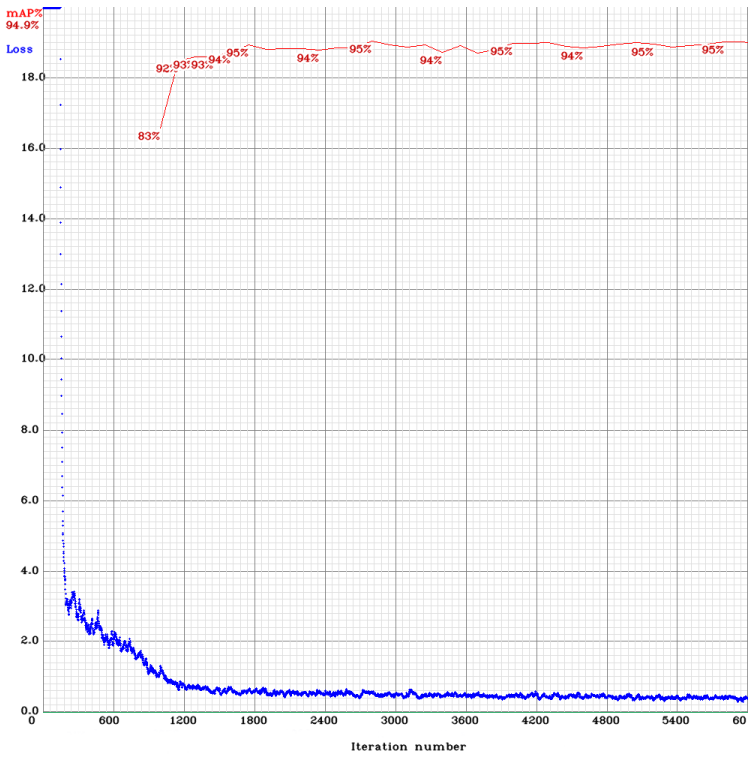

Figure 4.10: Average loss (blue) and mAP (red) curve for YOLOv4-baseline.

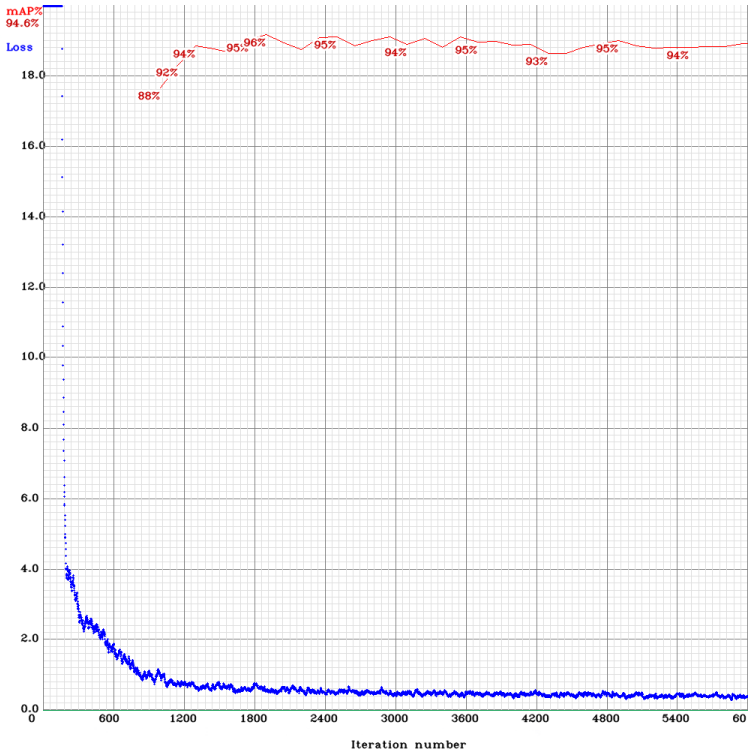

Figure 4.11: Average loss (blue) and mAP (red) curve for YOLOv4-SAM-Mish.

4.3.2 Evaluation

Table 4.1 shows the mAP@0.5 on the validation set for both models. Both models show high mAP@0.5. However, surprisingly the YOLOv4-baseline model achieved a slightly better result than the YOLOv4-SAM-Mish model. Even though Bochkovskiy et al. [39] reported better performance on MS COCO with the additional SAM block and Mish activation on the neck, we think that our simple one-class dataset might not need those additional techniques to achieve good performance.

Model	**mAP@0.5**
YOLOv4-baseline	94.93%
YOLOv4-SAM-Mish	94.56%

Table 4.1: mAP@0.5 for YOLOv4-baseline and YOLOv4-SAM-Mish.

We also compared True Positives (TP), False Positives (FP), and False Negatives (FN) at different confidence thresholds for both models, demonstrated in Table 4.2. We can also see that YOLOv4-baseline gives overall slightly more accurate predictions: The TP predictions (count of predicted boxes correctly including corals) are slightly higher for YOLOv4-baseline at all confidence thresholds. Furthermore, the FN predictions (count of undetected corals) are a little bit lower for YOLOv4-baseline at all confidence thresholds. Finally, the FP predictions (count of objects wrongly predicted as corals) are slightly lower for YOLOv4-baseline at the confidence

threshold of 0.75. However, the FP predictions at the confidence thresholds 0.25 and 0.5 are rather higher for YOLOv4-baseline.

Confidence Threshold 0.25			
Model	**TP**	**FP**	**FN**
YOLOv4-baseline	313	31	28
YOLOv4-SAM-Mish	307	28	34
Confidence Threshold 0.5			
Model	**TP**	**FP**	**FN**
YOLOv4-baseline	294	19	47
YOLOv4-SAM-Mish	292	18	49
Confidence Threshold 0.75			
Model	**TP**	**FP**	**FN**
YOLOv4-baseline	280	5	61
YOLOv4-SAM-Mish	277	7	64

Table 4.2: True Positives (TP), False Positives (FP) and False Negatives (FN) at different confidence thresholds for both models.

To conclude, the YOLOv4-baseline model got a marginally higher mAP@0.5, higher TP predictions, and lower FN predictions, except for the FP predictions, which are lower for the YOLOv4-SAM-Mish model at two of the three confidence thresholds. Taking all these observations into account, we think that YOLOv4-baseline showed to some degree better validation performance, and we, therefore, chose it as our final model.

4.3.3 Testing

Testing on data with the same environmental settings

The results from testing on data from the Kosterhavet National Park with the same environmental settings are shown in Table 4.3 and Table 4.4. Table 4.3 shows that our model achieved a higher mAP@0.5 (63.61% compared to 48.49%).
In Table 4.4, we computed TP, FP, FN, Precision and Recall for both models at three different confidence thresholds: 0.25, 0.5 and 0.75. Our model reached a higher Recall than the KSO model at all confidence thresholds, which indicates that our model succeeded in detecting more true corals. The second row in Figure 4.12 shows an example where our model (right) successfully detected the coral, while the KSO model (left) failed. However, the KSO model achieved higher Precision at all confidence thresholds demonstrating that their model made fewer miss-classifications. The first row in Figure 4.12 shows an example where our model (right) miss-classified the rosy feather star as a coral while the KSO model (left) succeeded in detecting only the coral.
Even though our model scored higher Recall than the KSO model, our model still resulted in a high number of FN predictions, which displays that many corals did not get detected at all.

Model	mAP@0.5
YOLOv4-baseline (ours)	63.61%
KSO model	48.49%

Table 4.3: mAP@0.5 for the KSO model and YOLOv4-baseline (ours) on the data from the Kosterhavet National Park with the same environmental settings.

Confidence Threshold 0.25					
Model	**TP**	**FP**	**FN**	**Precision**	**Recall**
YOLOv4-baseline (ours)	280	92	262	0.75	0.52
KSO model	214	51	328	0.81	0.39
Confidence Threshold 0.5					
Model	**TP**	**FP**	**FN**	**Precision**	**Recall**
YOLOv4-baseline (ours)	226	45	316	0.83	0.42
KSO model	154	13	388	0.92	0.28
Confidence Threshold 0.75					
Model	**TP**	**FP**	**FN**	**Precision**	**Recall**
YOLOv4-baseline (ours)	164	16	378	0.91	0.30
KSO model	90	5	452	0.95	0.17

Table 4.4: TP, FP, FN, Precision and Recall at different confidence thresholds for the KSO model and YOLOv4-baseline (ours) on data from the Kosterhavet National Park with the same environmental settings.

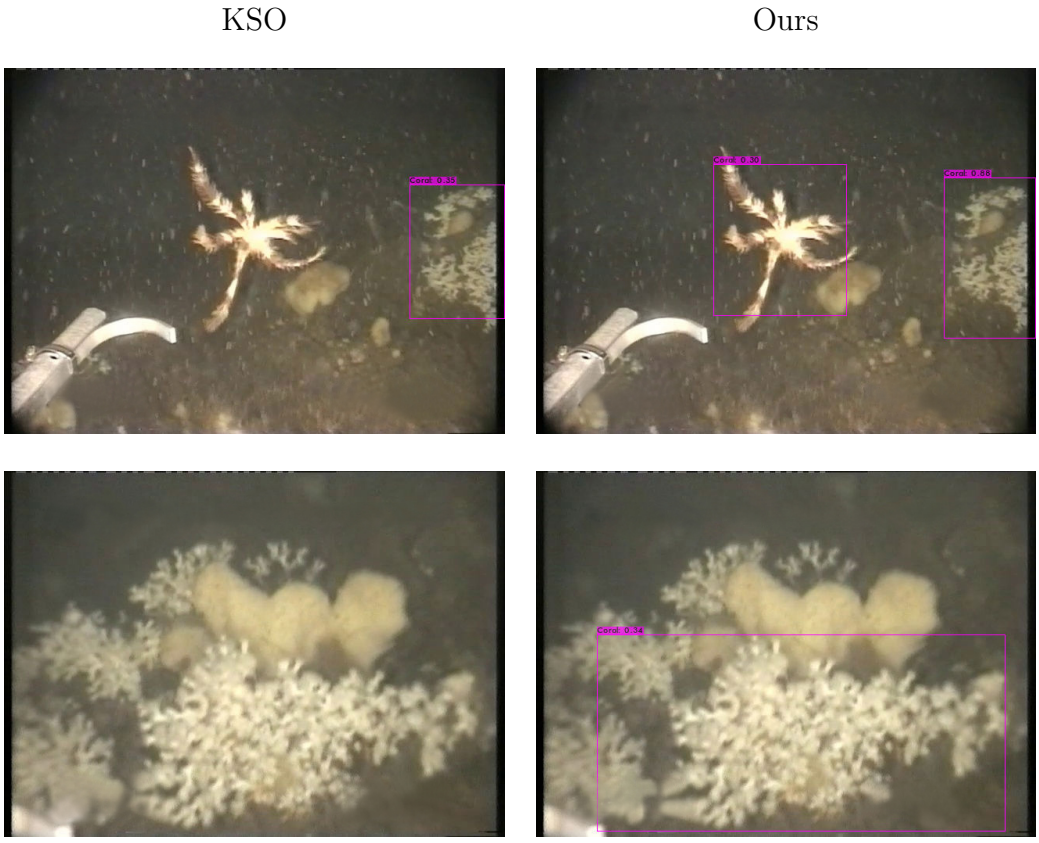

Figure 4.12: Example images with predicted boxes. The KSO model predictions to the left. Our model predictions to the right.

Testing on data with different environmental settings

When testing on the Kosterhavet footages, which were recorded by a newer camera and hence held different environmental settings, both models achieved rather low mAP@0.5. However, our model still scored a higher mAP@0.5 (27.93% compared to 13.74%), see Table 4.6. In Table 4.5, we compared TP, FP, FN, Precision and Recall for both models at different confidence thresholds. Throughout all thresholds, we observed that our model got higher Precision compared to the KSO model, meaning that the number of true detected corals over the detected corals is higher for our model. Also, our model got higher Recall compared to the KSO model, demonstrating that the fraction of true detected corals among all true corals is higher for our model. However, our model has a high number of FP predictions for all confidence thresholds, meaning that our model makes many miss-classifications. For example, the first row in Figure 4.13 shows a frame where our model predicted a Bolocera wrongly as a coral, while the KSO model behaved in this situation correctly. Another interesting observation is that when increasing the confidence threshold, the number of detections of the KSO model dropped notably in contrast to our model, which means that most of the confidence scores for the KSO model are not too high. The second row in Figure 4.13 shows an example where our model detected the coral with a confidence score of 0.91, whereas the KSO model's detection had a confidence score of 0.47. Lastly, we noticed that many corals got detected by neither the KSO model nor our model. This is indicated by the high number of FN predictions, which is more than half of the ground truth boxes.

Confidence Threshold 0.25					
Model	**TP**	**FP**	**FN**	**Precision**	**Recall**
YOLOv4-baseline (ours)	27	59	29	0.31	0.48
KSO model	18	45	38	0.29	0.32
Confidence Threshold 0.5					
Model	**TP**	**FP**	**FN**	**Precision**	**Recall**
YOLOv4-baseline (ours)	21	35	35	0.38	0.38
KSO model	8	16	48	0.33	0.14
Confidence Threshold 0.75					
Model	**TP**	**FP**	**FN**	**Precision**	**Recall**
YOLOv4-baseline (ours)	16	22	40	0.42	0.29
KSO model	1	3	55	0.25	0.02

Table 4.5: TP, FP, FN, Precision, and Recall at different confidence thresholds for the KSO model and YOLOv4-baseline (ours) on the data from the Kosterhavet National Park with different environmental settings.

Model	**mAP@0.5**
YOLOv4-baseline (ours)	27.93%
KSO model	13.74%

Table 4.6: mAP@0.5 for the KSO model and YOLOv4-baseline (ours) on the data from the Kosterhavet National Park with different environmental settings.

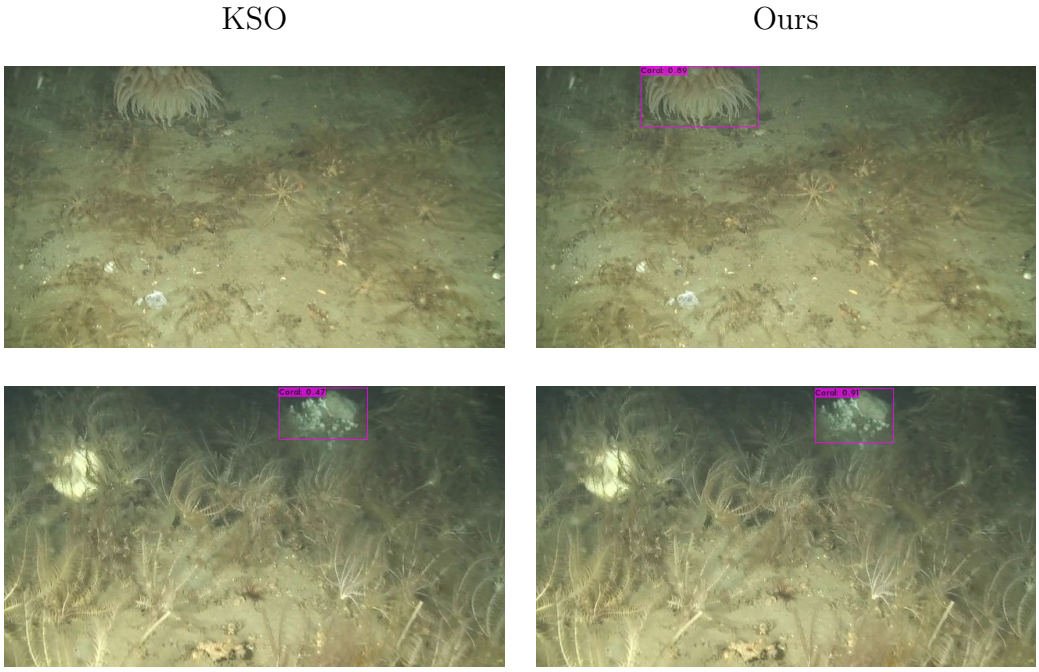

Figure 4.13: Example images with predicted boxes for the data with different environmental settings.

Testing on data from outside the Kosterhavet National Park

Both models resulted in low mAP@0.5 on the MMT-test data. However, our model still achieved a higher mAP@0.5 than the KSO model, i.e., 32.01% compared to 10.13%, see Table 4.7. Table 4.8 shows the results of TP, FP, FN, Precision and Recall for both models at three different confidence thresholds. Both models have a very high number of FN predictions and a low number of TP and FP predictions, meaning that many corals did not get detected by the models. For example, the first row in Figure 4.7 shows an example where both models missed to detect the coral. However, the results indicate that our model performed better than the KSO model. For example, at confidence threshold 0.25, our model correctly detected 18 corals compared to the KSO model that only detected 5 corals out of 100. An example of this can also be seen in the second row in Figure 4.7, where our model detected the coral, while the KSO model failed.

Model	**mAP@0.5**
YOLOv4-baseline (ours)	32.01%
KSO model	10.13%

Table 4.7: mAP@0.5 for the KSO model and YOLOv4-baseline (ours) on the MMT-test data.

Confidence Threshold 0.25					
Model	**TP**	**FP**	**FN**	**Precision**	**Recall**
YOLOv4-baseline (ours)	18	6	82	0.75	0.18
KSO model	5	10	95	0.33	0.05
Confidence Threshold 0.5					
Model	**TP**	**FP**	**FN**	**Precision**	**Recall**
YOLOv4-baseline (ours)	9	4	91	0.69	0.09
KSO model	1	4	99	0.20	0.01
Confidence Threshold 0.75					
Model	**TP**	**FP**	**FN**	**Precision**	**Recall**
YOLOv4-baseline (ours)	6	3	94	0.67	0.06
KSO model	0	1	100	0.00	0.00

Table 4.8: TP, FP, FN, Precision, and Recall at different confidence thresholds for the KSO model and YOLOv4-baseline (ours) on the MMT-test data.

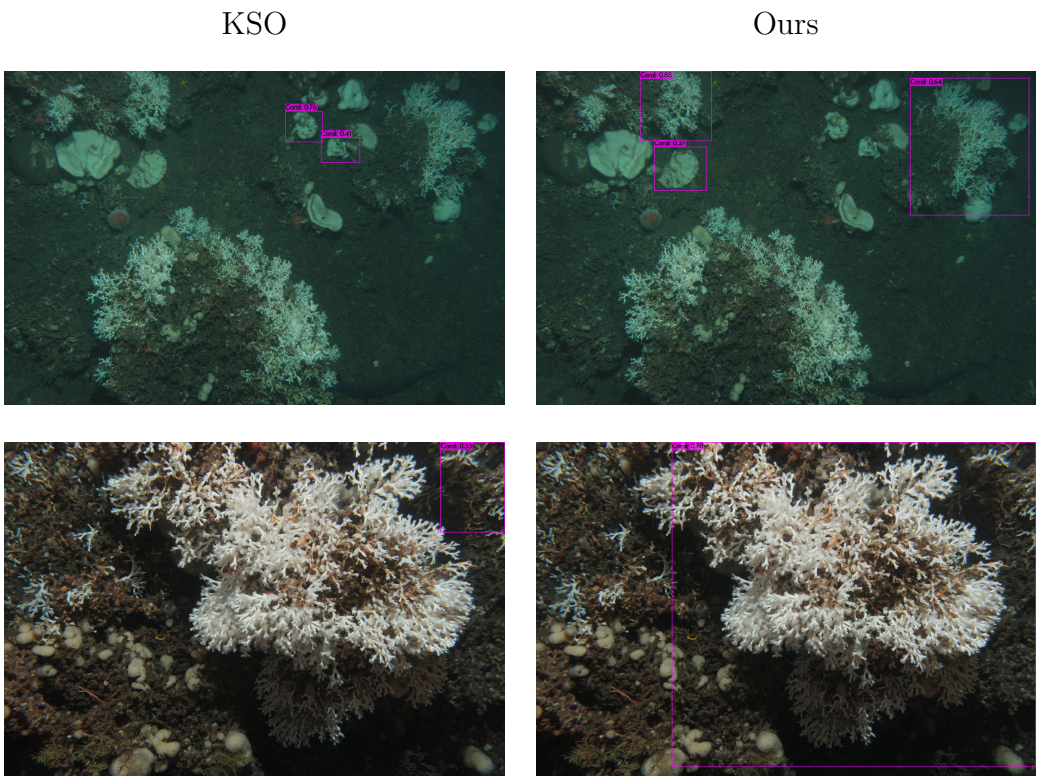

Figure 4.14: Example images with predicted boxes for the MMT data with different environmental settings outside the Kosterhavet National Park.

According to the test results in the three different experiments, we believe that our model achieved promising initial results that can be built upon in the future. The results from testing on data from the Kosterhavet National Park similar to the training data show that our model outperforms the KSO model with a 15% higher mAP@0.5, which indicates that our proposed system has the potential to overcome the overfitting issue suffered by the KSO baseline model. Moreover, the tests on

data from different environments, inside and outside the Kosterhavet National Park, demonstrated that our proposed system generalizes better to new domains than the KSO model. However, the resulted scores from testing on data from different environments are not very high. The reason is that our system resulted in high numbers of False Positive and False Negative predictions, which means that the system makes many miss-classifications and misses to detect some of the corals. A reason for this could be that our training data includes only 2675 images (409 real images and 2266 synthetic images), which might not be diverse enough to cover all cases and generalize well on different environments. Also, we did not train the system to learn what is not a coral or to distinguish between different species. This can be done, for example, by including negative examples, images that contain different objects other than corals [65]. Training with negative examples might improve the learning process, make the system more robust and reduce the number of False Positive predictions. Moreover, we think that one cause for the high number of undetected corals is that many corals are hard to detect due to the poor quality of the underwater footage. Unfortunately, we could not finish building the enhancement pipeline to improve the images by making the coral's features more visible, which might result in making the detection easier and hence, reduce the False Negative predictions.

We computed TP, FP, FN, Precision, and Recall at different confidence thresholds to analyze the influence of the threshold on the resulting detections. In all three tests we observed a rather linear relationship between the confidence thresholds and the computed metrics (see Table 4.8, 4.5 and 4.4). This implicates that there is no clear threshold giving better results than the others.

The results showcase that our proposed model is superior to the KSO model in all three analyzed scenarios in terms of mAP@0.5. However, one emerging question is what role StyleGAN2 in the data augmentation process plays, besides the change of using YOLOv4 instead of YOLOv3. The authors of YOLOv4 reported an increase of about 10% in mAP@0.5 compared to YOLOv3, both trained on MS COCO [39]. Our model was trained on fewer data compared to the KSO-model (YOLOv3) yet achieved an increase between 13-21% mAP@0.5 through the conducted tests. Therefore, we believe that not only the replacement of YOLOv3 with YOLOv4 was crucial to the improvement in performance, but also the introduction of synthetic data. However, to precisely quantify how much the synthetic data contributed to the success, further testing is needed.

In the case of rare species, like the deepwater corals, where gathering enough diverse data to train an object detector and achieve desired evaluation metrics is hard, we believe that using synthetic data is useful. Furthermore, in the case of multi-class detection, GANs could also be useful when the training data is unbalanced. However, in the case of species where a lot of diverse data is available, it has to be tested how the combination of synthetic and real data would perform against only real data. Moreover, the time needed to prepare thousands of real images (i.e., cutting footage, finding frames with the desired species) versus the time needed to prepare only a few hundred real images and then increase it synthetically using a GAN, has to be considered during the comparison.

To sum up, we believe that our results are a promising start, and more future research can be done to further develop the system to become more robust and generalize well on new and different environments.

5

Conclusion

In this thesis, we explored building a more robust system to improve the detection accuracy for deepwater corals. Our research included three phases: image enhancement, image augmentation, and object detection. The results of each phase are summarized below.

Given the enhancement phase results, we have found that creating an effective and usable enhancement network using CycleGAN is a difficult task in our case. We believe that one reason for this is that we lack good quality images with the same characteristics as the Kosterhavet images to train CycleGAN successfully. The results stated in the report are the best result we got. Besides that, we also experimented with splitting the deeplet sea anemones dataset (domain X data from the CycleGAN training in Section 3.2) into bad and good quality images to get two datasets, which only differ in quality. However, the good images were not good enough to achieve an adequate enhancement system. To build an enhancement system using CycleGAN, we believe that good and bad quality images from the same region are needed. We also want to mention that several authors focused on building advanced and complex algorithms, like UGAN [66], which might give better enhancement results for underwater images than CycleGAN. However, UGAN also requires two sets of good and bad quality images, making it impractical to use at this stage. Therefore, we leave the research about building an enhancement network for the KSO pipeline for future work.

The results from the augmentation phase showed that GANs could be used to increase the training data. All three tested models produced diverse synthetic data, which showed the characteristics of the training data. However, the first trained networks, DCGAN and WGAN-GP, have the limitation that they can only generate images of size 64×64. Therefore, we used the more advanced model StyleGAN2 to output 2266 images of resolution 512×512 to train the object detector.

In the last step, we trained two YOLOv4 based object detection systems, YOLOv4-baseline and YOLOv4-SAM-Mish. Both models showed satisfactory results on the evaluation dataset. However, the YOLOv4-baseline model gave slightly better prediction performance and was therefore chosen as our final model to test against the KSO-model and answer our research questions. Our system achieved fairly good results on the testing data from the Kosterhavet National Park with the same environmental settings and outperformed the KSO model in terms of mAP@0.5. Therefore, we believe that our proposed system came a step closer to overcome the overfitting issue. In terms of generalization towards different environments inside as

well as outside the Kosterhavet National Park, our proposed system also achieved better results than the baseline model implemented in the KSO system. However, the results still show that more research has to be done towards building a system that can successfully generalize to new domains. This because in all three tests, we observed high numbers of False Negative and False Positive predictions. Therefore, further research into building a more robust system to make it easier for the detector to recognize the deepwater corals, distinguish between different species, and generalize better to new and different environments is needed.

6
Future Work

Data Enhancement

To further investigate how to build an effective enhancement system using CycleGAN or UGAN, we believe that a set of good quality images from the Kosterhavet National Park has to be collected. Training the image translation network with two sets of images that only differ in terms of quality might result in an adequate enhancement system.

Data Augmentation

Interesting future research about using GANs for data augmentation could include further investigation of increasing the data diversity. We think that this could be done by exploring more image-to-image translation methods to change an image background without affecting objects of interest. Also, training StyleGAN2 with images that include more species might result in creating synthetic images, which are more diverse.
Furthermore, due to time constraints and long training hours, we only trained StyleGAN2 with the default hyperparameters suggested by the authors. Tuning some hyperparameters in StyleGAN2 might enhance the quality of images, generate more realistic data, and improve the overall performance of the system.
Finally, training with a large dataset needs a long time to train and a lot of RAM. This was one of the reasons why we only generated 3000 synthetic images. Hence, we leave the generation of more synthetic data and the investigation of how this affects the model performance for future work.

Object detection

Further research concerning the improvement of the coral detection system could include adding negative samples to the training data [65]. Negative samples are unlabeled images, which include other objects besides corals (i.e., fishes, sponges, etc.). This might further improve the detection performance since the detector not only learns the features of corals but also learns to distinguish between corals and other objects. Furthermore, due to time constraints, we only trained the YOLOv4-baseline and YOLOv4-SAM-Mish models with the default hyperparameters. Further experiments to adjust the hyperparameters might boost the detection performance. Finally, we also think that adding more traditional data augmentation techniques, like randomly flipping and rotating the images, could be investigated to see if it increases the performance of the model.

Finally, the appropriate metric for the model's evaluation should be carefully chosen depending on the employment of the system. In case the system is used as a

recommendation system, Recall should be prioritized. However, for a stand-alone system, both Precision and Recall should be considered.

Explore more species

An exciting part of future research would be trying to make our system recognize more species. Important species that could be explored in the Kosterhavet are habitat builders, including *Sponges* (Axinella spp., Phakelia ventilabrum, Geodia baretti), *Sea pens/anemones* (Kophobelemnon stelliferum, Pennatula phosphorea, Bolocera tuediae) and *Crustaceans* (Nephrops norvegicus).

Bibliography

[1] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and F. Li, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*. IEEE Computer Society, 2009, pp. 248–255. [Online]. Available: https://doi.org/10.1109/CVPR.2009.5206848

[2] O. Matthias. (2020, Apr. 29) How machines change marine biology. Accessed Jan. 18, 2021. [Online]. Available: https://www.youtube.com/watch?v=wxg3w3VLIbM&t=2s

[3] L. Guidi, A. Fernàndez-Guerra, D. Bakker, C. Canchaya, E. Curry, F. Foglini, J.-O. Irission, K. Malde, C. T. Marshall, M. Obst, R. P. Ribeiro, and J. Tjiputra, "Big data in marine science." Zenodo, 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3755793

[4] F. Sultana, A. Sufian, and P. Dutta, "Advancements in image classification using convolutional neural network," *CoRR*, vol. abs/1905.03288, 2019. [Online]. Available: http://arxiv.org/abs/1905.03288

[5] Z. Zhao, P. Zheng, S. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE Trans. Neural Networks Learn. Syst.*, vol. 30, no. 11, pp. 3212–3232, 2019. [Online]. Available: https://doi.org/10.1109/TNNLS.2018.2876865

[6] C. Fabbri, M. J. Islam, and J. Sattar, "Enhancing underwater imagery using generative adversarial networks," in *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*. IEEE, 2018, pp. 7159–7165. [Online]. Available: https://doi.org/10.1109/ICRA.2018.8460552

[7] V. Anton, J. Germishuys, P. Bergström, M. Lindegarth, and M. Obst, "An open-source, citizen science and machine learning approach to analyse subsea movies," *Biodiversity Data Journal*, vol. 9, p. e60548, 2021. [Online]. Available: https://doi.org/10.3897/BDJ.9.e60548

[8] S. Mallick. (2017, Feb. 13) Object tracking using opencv (c++/python). Accessed Jan. 20, 2021. [Online]. Available: https://learnopencv.com/object-tracking-using-opencv-cpp-python/

[9] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *CoRR*, vol. abs/1804.02767, 2018. [Online]. Available: http://arxiv.org/abs/1804.02767

[10] M. Z. Alom, T. M. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. S. Nasrin, M. Hasan, B. C. Van Essen, A. A. S. Awwal, and V. K. Asari, "A state-of-the-art survey on deep learning theory and architectures," *Electronics*, vol. 8, no. 3, 2019. [Online]. Available: https://www.mdpi.com/2079-9292/8/3/292

[11] M. Frid-Adar, I. Diamant, E. Klang, M. Amitai, J. Goldberger, and H. Greenspan, "Gan-based synthetic medical image augmentation for increased CNN performance in liver lesion classification," *CoRR*, vol. abs/1803.01229, 2018. [Online]. Available: http://arxiv.org/abs/1803.01229

[12] O. Boulais, B. Woodward, B. Schlining, L. Lundsten, K. Barnard, K. C. Bell, and K. Katija, "Fathomnet: An underwater image training database for ocean exploration and discovery," *arXiv e-prints*, 2020. [Online]. Available: https://arxiv.org/abs/2007.00114

[13] E. Ditria, M. Sievers, S. Lopez-Marcano, E. L. Jinks, and R. M. Connolly, "Deep learning for automated analysis of fish abundance: the benefits of training across multiple habitats," *bioRxiv*, 2020. [Online]. Available: https://www.biorxiv.org/content/early/2020/05/22/2020.05.19.105056

[14] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, "Mask R-CNN," in *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*. IEEE Computer Society, 2017, pp. 2980–2988. [Online]. Available: https://doi.org/10.1109/ICCV.2017.322

[15] V. López-Vázquez, J. M. López-Guede, S. Marini, E. Fanelli, E. Johnsen, and J. Aguzzi, "Video image enhancement and machine learning pipeline for underwater animal detection and classification at cabled observatories," *Sensors*, vol. 20, no. 3, p. 726, 2020. [Online]. Available: https://doi.org/10.3390/s20030726

[16] A. M. Reza, "Realization of the contrast limited adaptive histogram equalization (CLAHE) for real-time image enhancement," *J. VLSI Signal Process.*, vol. 38, no. 1, pp. 35–44, 2004. [Online]. Available: https://doi.org/10.1023/B:VLSI.0000028532.53893.82

[17] S. Song, J. Zhu, X. Li, and Q. Huang, "Integrate MSRCR and mask R-CNN to recognize underwater creatures on small sample datasets," *IEEE Access*, vol. 8, pp. 172 848–172 858, 2020. [Online]. Available: https://doi.org/10.1109/ACCESS.2020.3025617

[18] T. R. Shaham, T. Dekel, and T. Michaeli, "Singan: Learning a generative model from a single natural image," in *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*. IEEE, 2019, pp. 4569–4579. [Online]. Available: https://doi.org/10.1109/ICCV.2019.00467

[19] Sjöfartsverket. (2019, Jul. 19) Spridningstillstånd. Accessed Jan. 22, 2021. [Online]. Available: https://www.sjofartsverket.se/sv/Batliv/Sjokort/Copyright--nyttjanderatt/Spridningstillstand/

[20] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, Eds., vol. 27. Curran Associates, Inc., 2014, pp. 2672–2680. [Online]. Available: https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf

[21] Z. Pan, W. Yu, X. Yi, A. Khan, F. Yuan, and Y. Zheng, "Recent progress on generative adversarial networks (gans): A survey," *IEEE Access*, vol. 7, pp. 36 322–36 333, 2019. [Online]. Available: https://doi.org/10.1109/ACCESS.2019.2905015

[22] ——, "Recent progress on generative adversarial networks (gans): A survey," *IEEE Access*, vol. 7, pp. 36 322–36 333, 2019. [Online]. Available: https://doi.org/10.1109/ACCESS.2019.2905015

[23] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2016. [Online]. Available: http://arxiv.org/abs/1511.06434

[24] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein GAN," *CoRR*, vol. abs/1701.07875, 2017. [Online]. Available: http://arxiv.org/abs/1701.07875

[25] C. Villani, *Optimal Transport: Old and New*, ser. Grundlehren der mathematischen Wissenschaften. Springer, Berlin, Heidelberg, 2009.

[26] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein gans," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, Eds., 2017, pp. 5767–5777. [Online]. Available: https://proceedings.neurips.cc/paper/2017/hash/892c3b1c6dccd52936e27cbd0ff683d6-Abstract.html

[27] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.6980

[28] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 2019, pp. 4401–4410. [Online]. Available: http://openaccess.thecvf.com/content_CVPR_2019/html/Karras_A_Style-Based_Generator_Architecture_for_Generative_Adversarial_Networks_CVPR_2019_paper.html

[29] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive growing of gans

for improved quality, stability, and variation," in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. [Online]. Available: https://openreview.net/forum?id=Hk99zCeAb

[30] X. Huang and S. J. Belongie, "Arbitrary style transfer in real-time with adaptive instance normalization," in *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*. IEEE Computer Society, 2017, pp. 1510–1519. [Online]. Available: https://doi.org/10.1109/ICCV.2017.167

[31] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila, "Analyzing and improving the image quality of stylegan," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*. IEEE, 2020, pp. 8107–8116. [Online]. Available: https://doi.org/10.1109/CVPR42600.2020.00813

[32] J. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*. IEEE Computer Society, 2017, pp. 2242–2251. [Online]. Available: https://doi.org/10.1109/ICCV.2017.244

[33] J. Johnson, A. Alahi, and L. Fei-Fei, "Perceptual losses for real-time style transfer and super-resolution," in *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part II*, ser. Lecture Notes in Computer Science, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds., vol. 9906. Springer, 2016, pp. 694–711. [Online]. Available: https://doi.org/10.1007/978-3-319-46475-6_43

[34] D. Ulyanov, A. Vedaldi, and V. S. Lempitsky, "Instance normalization: The missing ingredient for fast stylization," *CoRR*, vol. abs/1607.08022, 2016. [Online]. Available: http://arxiv.org/abs/1607.08022

[35] P. Isola, J. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*. IEEE Computer Society, 2017, pp. 5967–5976. [Online]. Available: https://doi.org/10.1109/CVPR.2017.632

[36] S. Zhao, Z. Liu, J. Lin, J. Zhu, and S. Han, "Differentiable augmentation for data-efficient GAN training," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., 2020. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/55479c55ebd1efd3ff125f1337100388-Abstract.html

[37] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *2014*

IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014. IEEE Computer Society, 2014, pp. 580–587. [Online]. Available: https://doi.org/10.1109/CVPR.2014.81

[38] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2016, pp. 779–788. [Online]. Available: https://doi.org/10.1109/CVPR.2016.91

[39] A. Bochkovskiy, C. Wang, and H. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *CoRR*, vol. abs/2004.10934, 2020. [Online]. Available: https://arxiv.org/abs/2004.10934

[40] J. Redmon and A. Farhadi, "YOLO9000: better, faster, stronger," in *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*. IEEE Computer Society, 2017, pp. 6517–6525. [Online]. Available: https://doi.org/10.1109/CVPR.2017.690

[41] T. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie, "Feature pyramid networks for object detection," in *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*. IEEE Computer Society, 2017, pp. 936–944. [Online]. Available: https://doi.org/10.1109/CVPR.2017.106

[42] T. Lin, M. Maire, S. J. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: common objects in context," in *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V*, ser. Lecture Notes in Computer Science, D. J. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds., vol. 8693. Springer, 2014, pp. 740–755. [Online]. Available: https://doi.org/10.1007/978-3-319-10602-1_48

[43] C. Wang, H. M. Liao, Y. Wu, P. Chen, J. Hsieh, and I. Yeh, "Cspnet: A new backbone that can enhance learning capability of CNN," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR Workshops 2020, Seattle, WA, USA, June 14-19, 2020*. IEEE, 2020, pp. 1571–1580. [Online]. Available: https://doi.org/10.1109/CVPRW50498.2020.00203

[44] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, "Path aggregation network for instance segmentation," *CoRR*, vol. abs/1803.01534, 2018. [Online]. Available: http://arxiv.org/abs/1803.01534

[45] S. Yun, D. Han, S. Chun, S. J. Oh, Y. Yoo, and J. Choe, "Cutmix: Regularization strategy to train strong classifiers with localizable features," in *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*. IEEE, 2019, pp. 6022–6031. [Online]. Available: https://doi.org/10.1109/ICCV.2019.00612

[46] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *2016 IEEE Conference on*

Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016. IEEE Computer Society, 2016, pp. 2818–2826. [Online]. Available: https://doi.org/10.1109/CVPR.2016.308

[47] Z. Zheng, P. Wang, W. Liu, J. Li, R. Ye, and D. Ren, "Distance-iou loss: Faster and better learning for bounding box regression," in *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*. AAAI Press, 2020, pp. 12 993–13 000. [Online]. Available: https://aaai.org/ojs/index.php/AAAI/article/view/6999

[48] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, ser. JMLR Workshop and Conference Proceedings, F. R. Bach and D. M. Blei, Eds., vol. 37. JMLR.org, 2015, pp. 448–456. [Online]. Available: http://proceedings.mlr.press/v37/ioffe15.html

[49] D. Misra, "Mish: A self regularized non-monotonic activation function," in *31st British Machine Vision Conference 2020, BMVC 2020, Virtual Event, UK, September 7-10, 2020*. BMVA Press, 2020. [Online]. Available: https://www.bmvc2020-conference.com/assets/papers/0928.pdf

[50] S. Woo, J. Park, J. Lee, and I. S. Kweon, "CBAM: convolutional block attention module," in *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part VII*, ser. Lecture Notes in Computer Science, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds., vol. 11211. Springer, 2018, pp. 3–19. [Online]. Available: https://doi.org/10.1007/978-3-030-01234-2_1

[51] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 9, pp. 1904–1916, 2015. [Online]. Available: https://doi.org/10.1109/TPAMI.2015.2389824

[52] Z. Huang, J. Wang, X. Fu, T. Yu, Y. Guo, and R. Wang, "DC-SPP-YOLO: dense connection and spatial pyramid pooling based YOLO for object detection," *Inf. Sci.*, vol. 522, pp. 241–258, 2020. [Online]. Available: https://doi.org/10.1016/j.ins.2020.02.067

[53] M. Everingham, L. V. Gool, C. K. I. Williams, J. M. Winn, and A. Zisserman, "The pascal visual object classes (VOC) challenge," *Int. J. Comput. Vis.*, vol. 88, no. 2, pp. 303–338, 2010. [Online]. Available: https://doi.org/10.1007/s11263-009-0275-4

[54] J. Yu, Y. Jiang, Z. Wang, Z. Cao, and T. S. Huang, "Unitbox: An advanced object detection network," in *Proceedings of the 2016 ACM Conference on Multimedia Conference, MM 2016, Amsterdam, The Netherlands, October 15-19, 2016*, A. Hanjalic, C. Snoek, M. Worring, D. C. A. Bulterman, B. Huet,

A. Kelliher, Y. Kompatsiaris, and J. Li, Eds. ACM, 2016, pp. 516–520. [Online]. Available: https://doi.org/10.1145/2964284.2967274

[55] F. Bashir and F. Porikli, "Performance evaluation of object detection and tracking systems," Mitsubishi Electric Research Laboratories, Tech. Rep., Jun. 2006. [Online]. Available: http://www.merl.com/reports/docs/TR2006-041.pdf

[56] Wikipedia contributors, "Precision and recall — Wikipedia, the free encyclopedia," 2021, accessed May. 03, 2021. [Online]. Available: https://en.wikipedia.org/wiki/Precision_and_recall

[57] X. Mao, Q. Li, H. Xie, R. Y. K. Lau, Z. Wang, and S. P. Smolley, "Least squares generative adversarial networks," in *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*. IEEE Computer Society, 2017, pp. 2813–2821. [Online]. Available: https://doi.org/10.1109/ICCV.2017.304

[58] T. Salimans, I. J. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," *CoRR*, vol. abs/1606.03498, 2016. [Online]. Available: http://arxiv.org/abs/1606.03498

[59] N. Inkawhich. (2017) Dcgan tutorial. Accessed Mar. 10, 2021. [Online]. Available: https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html

[60] A. Persson. (2021) Machine learning collection. Accessed Mar. 10, 2021. [Online]. Available: https://github.com/aladdinpersson/Machine-Learning-Collection/tree/master/ML/Pytorch/GANs/4.%20WGAN-GP

[61] L. M. Mescheder, A. Geiger, and S. Nowozin, "Which training methods for gans do actually converge?" in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, ser. Proceedings of Machine Learning Research, J. G. Dy and A. Krause, Eds., vol. 80. PMLR, 2018, pp. 3478–3487. [Online]. Available: http://proceedings.mlr.press/v80/mescheder18a.html

[62] Z. Si and S. Zhu, "Learning hybrid image templates (HIT) by information projection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 7, pp. 1354–1367, 2012. [Online]. Available: https://doi.org/10.1109/TPAMI.2011.227

[63] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, Eds., 2017, pp. 6626–6637. [Online]. Available: https://proceedings.neurips.cc/paper/2017/hash/8a1d694707eb0fefe65871369074926d-Abstract.html

[64] D. Tzutalin. (2015) LabelImg: A graphical image annotation tool. Accessed Mar. 19, 2021. [Online]. Available: https://github.com/tzutalin/labelImg

[65] J. Redmon, "Darknet: Open source neural networks in c," http://pjreddie.com/darknet/, 2013–2016.

[66] C. Fabbri, M. J. Islam, and J. Sattar, "Enhancing underwater imagery using generative adversarial networks," in *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*. IEEE, 2018, pp. 7159–7165. [Online]. Available: https://doi.org/10.1109/ICRA.2018.8460552

A

Appendix 1

A.1 Data

Name	Description	Quantity	Source	Licence
Coral Frames	Our initial training frames containing deepwater corals taken by ROVs in the Kosterhavet National Park.	409	KSO	CC BY 4.0.
Bolocera Frames	Bolocera images used as Domain X in the enhancement process. Taken by ROVs in the Kosterhavet.	409	KSO	CC BY 4.0.
Old Movie	Footage used to test the ability of our model to overcome the overfitting issue on data similar to the training data. Taken by ROVs in the Kosterhavet.	1	KSO	CC BY 4.0.
New Movies	Testing movies taken by a newer camera model to investigate the generalization of our model to different environmental settings inside the Kosterhavet.	3	KSO	CC BY 4.0.
MMT Images	High resolution images taken outside the Kosterhavet National Park. The data was used together with the ImageNet images as domain Y during the training of Cyclegan. This dataset was also used to test the model's ability to generalize to new domains outside the Kosterhavet.	31	MMT Sweden AB	Restricted access
ImageNet Images	Hand picked images from ImageNet including the classes sea anemone and coral. These images formed together with the MMT Images the domain Y for Cyclegan training.	273	ImageNet	We refer to ImageNet's data licence page here.

Table A.1: Information about the used data in this study.

Table A.1 gives an overview of the data used in this study and provides information including data license and access. The main material was provided by the Koster

Seafloor Observatory (KSO) and can be accessed by contacting the responsible person displayed on their website. The images from MMT Sweden AB, that were used during the training of CycleGAN and the testing phase, have restricted access. We also used images from ImageNet during the training of CycleGAN, which can be downloaded under their given licence on their website. Finally, we also produced 2266 synthetic coral images, which will be catalogued in the Swedish National Data Service (SND) under the licence CC BY 4.0..