HELPING BIOLOGISTS FIND WHALES: AI-IN-THE-LOOP SUPPORT FOR ENVIRONMENTAL DATASET CREATION

by

Mirerfan Gheibi

Submitted in partial fulfillment of the requirements
for the degree of Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
November 2021

I dedicate this to my family and the love of my life, and the ones whom my little successes put big smiles on their faces ...

Table of Contents

List of Tables

List of Figures

Abstract

We develop a computer vision system to help biologists detect endangered whales. Given access to a limited dataset of aerial imagery (1544 images of mainly water), we implemented object detection and semantic segmentation models. For segmentation, we leverage the extreme data imbalance by introducing an elliptic annotation mechanism mitigating the need for tight annotations while still constrained by expert annotators' available time. Data scarcity made zero-false-negative rate infeasible, so we minimized false negatives while having few enough false positives that it could still help an expert annotator accelerate the annotation process itself. This would allow a bootstrapping dataset creation approach: collecting increasingly larger datasets in parallel with training increasingly accurate models.

We evaluated performance for the downstream bootstrapping task with an AI-in-the-Loop experiment. Motivated by the expert user's workflow, this required developing a feature-based clustering visualization of the images. Our segmentation system admitted few false negatives and was more efficient than manually data collection alone. While the proposed approach cannot entirely solve the challenge of the extremely small dataset, it suggests that a slightly larger dataset (e.g. adding 100 whale images would double the relevant training set) may be sufficient to bootstrap the training and collection with effectively no false negatives.

List of Abbreviations Used

ASPP	Atrous Spatial Pyramid Pooling.
CNN	Convolutional Neural Network.
COCO	Common Objects in Context.
CRF	Conditional Random Field.
DFO	Fisheries and Oceans Canada.
DPM	Deformable Part Model.
FCN	Fully Convolutional Networks.
FPN	Feature Pyramid Networks.
GPU	Graphics Processing Unit.
ILSVRC	ImageNet Large Scale Visual Recognition Challenge.
IoU	Intersection over Union.
IUCN	International Union for Conservation of Nature.
PASCAL	Pattern Analysis, Statistical Modelling and Computational Learning.
R-CNN	Region Based Convolutional Neural Networks.
ResNet	Residual Neural Network.
RoIP	Region of Interest Pooling.
RPN	Region Proposal Network.

SPP	Spatial Pyramid Pooling.
SSD	Single Shot MultiBox Detector.
VOC	Visual Object Classes.
YOLO	You Only Look Once.

Acknowledgements

Like many research projects, this one is not a solo work as well. Many caring friends and colleagues helped me with this work. I want to thank them and show my respect for their efforts to make this thesis possible.

I consider myself a lucky person that I have had the chance to work with **Dr. Sageev Oore**. I have learned so much from him, and he guided me to make a better version of myself. He has been very supportive and caring. I would like to extend my deepest gratitude to him for everything he has done for me.

Working with **Chandramouli Shama Sastry** was a great pleasure, and I appreciate his time and brilliant ideas. I wish him the best and plenty of success.

I would like to thank **Dr. Thomas Trappenberg** for teaching me machine learning and allowing me to be the head TA of the Introduction to Experimental Robotics course. I also want to thank **Dr. Vlado Keselj** for his informative and engaging NLP course.

I am also grateful to **Olivia Pisano** for her huge help during this thesis. Her involvement with the project was stellar. Many thanks to **Dr. Boris Worm** for making this collaboration possible.

I'm deeply indebted to **Dr. Evangelos Milios** for the opportunity he gave me to join DeepSense. I want to thank **Jennifer LaPlante**, the executive director of DeepSense, who taught me how various aspects of projects seem and work outside academia and in the industry. During my studies, **Dr. Lu Yang**, **Dr. Jason Newport**, and **Dr. Geetika Bhatia** from DeepSense helped me greatly, and I appreciate their responsiveness. I would like to thank **Dr. Christopher Whidden** for his help, teaching, and mentoring me while he was at DeepSense.

Finally, I want to verbalize my profound gratitude to my family. They ushered me in my entire life with their love and encouragement. I am deeply appreciative to have **Gashin Ghazizadeh**. She has changed my life forever and filled my heart with everlasting love. She is my greatest supporter, and without her, I would not be near the one I am today.

This work was supported by **Mitacs**[1] through the **Mitacs Accelerate program**. In addition to Mitacs, this research was enabled in part by support provided by **Global Spatial Technology Solutions**[2] in the form of the aforementioned Mitacs Accelerate program. Computations were performed on the **DeepSense**[3] high-performance computing platform. DeepSense is funded by **ACOA**, the Province of **Nova Scotia**, the **Centre for Ocean Ventures and Entrepreneurship (COVE)**, **IBM Canada Ltd.**, and the **Ocean Frontier Institute (OFI)**. The dataset used in this research was provided by **Fisheries and Oceans Canada (DFO)**[4] in the form of unprocessed bellycam images under the title of **Aerial Survey Imagery Data**. I want to thank **Vector Institute**[5] for their support of my work as well.

[1] mitacs.ca
[2] gsts.ca
[3] deepsense.ca
[4] dfo-mpo.gc.ca
[5] vectorinstitute.ai

Chapter 1

Introduction

This work aims to help biology researchers locate, classify, and annotate whales in aerial imagery.

1.1 Motivation

North Atlantic Right Whale, *Eubalaena glacialis*, is one of the heavily endangered whales species that is filed as "critically endangered (cr)" species in the International Union for Conservation of Nature (IUCN), with the number of 200-250 mature individuals in their Jan 2020 assessment [10]. In 2017, twelve North Atlantic Right Whales were killed by ship accidents and fishing gear entanglements [11]. Since 2017, they have been facing an ongoing unusual mortality event (UME). Figure 1.1 is taken from NOAA[1] and shows a North Atlantic Right Whale that is damaged by vessel strikes that the propeller scars are visible. These damages can fatally harm the animals by breaking their spine and affecting their internal organs. This species often visits the coast of Nova Scotia. Because of the decreasing trend in its population, it is crucial to prevent future unfortunate deaths.

To take effective management measures to help the North Atlantic Right Whales restore their population, researchers systematically analyze their behavior, population estimate, and changes, which is called surveying. Surveying whales is crucial for assessing whales, their population, density, and health. For surveying, researchers use acoustic and imagery data acquired using vessels, airplanes, and satellites. Then they manually check them for the presence of whales. The data source that we used in this research is aerial imagery. An airplane with cameras mounted on it flies over the target region for studying and captures images over the water to acquire aerial imagery data. Then researchers go through images to find whales in them for surveying. Manually identifying animals in aerial ocean imagery requires considerable

[1] https://www.fisheries.noaa.gov/species/north-atlantic-right-whale#overview

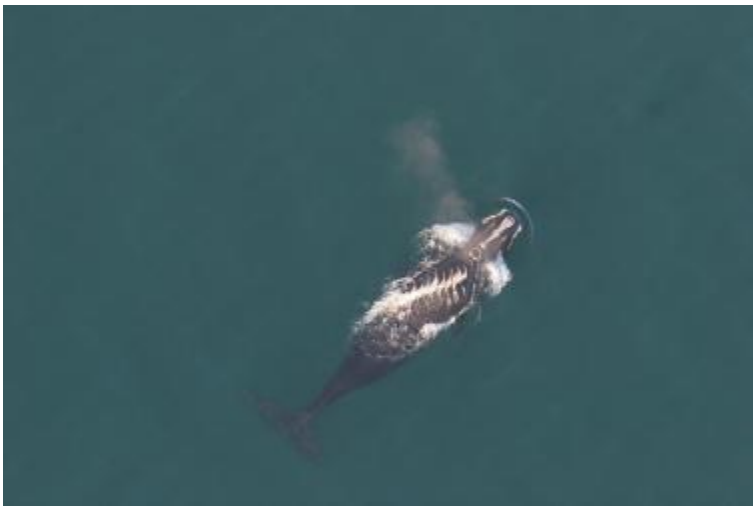


Figure 1.1: A Damaged North Atlantic Right Whale. Credit: NOAA

expertise and is very time-consuming. An automated system could facilitate this process, and that has been our aim in this work.

1.2 Problem Definition

Whales are rare, and because of the various depths that animals can swim, it is not common to capture them in aerial imagery. Therefore, the aerial survey imagery is mainly covered with water. This is the primary reason that makes finding whales, studying them, and collecting whale examples in aerial imagery highly time-consuming: the animals we are searching for account for only roughly 0.01% of the pixels in the available imagery.

Considering the need for an automatic census system to ease the manual checking efforts and the limited aerial imagery data we had for this project, we defined the objective of this research on two related goals. One of the goals is to develop a computer vision model to detect whales from aerial imagery. The other goal is to develop a whale detection system that is effective for downstream tasks. This system should be proficient to assist the bootstrap in the data collection process. These two goals are interconnected to each other; therefore, any progress in one of them is beneficial for the other.

As we will summarize shortly in the Overview (Section 1.4), we tried both object detection and semantic segmentation towards these objectives, and evaluated them

both individually and with a AI-in-the-loop experiment. Before giving this overview, we will first outline some of the challenges intrinsic to the problem we faced.

1.3 Challenges

The primary challenge faced when developing a computer vision model is data deficiency. Table 1.1 has some of the well-known benchmark datasets, with their number of examples, released for competitions or academic research to train and develop object detection models on them. The dataset we used in this research contains 1544 images, which is not adequate for training a deep architecture from scratch since the amount of needed data is immense for training deep architectures.

The data deficiency in this project has two interrelated causes. First, data collecting for a specific domain and task is relatively more difficult than collecting datasets of common objects. The difficulty is partially inherited from the number of instances of a particular object. For example, finding and collecting endangered whale images in aerial imagery is challenging due to their low number of instances in the world. Second, finding whales in the available imagery is laborious for human experts. The imagery usually contains an image from a region without any whales. However, because of the high resolution of the imagery, it takes extensive zooming and panning through a single large image to make sure there is indeed no animal, or to find the animal that might be there. Sometimes because of occlusion or other adverse effects, correctly identifying animals from this imagery needs discussion between researchers. There are 168 whales in our dataset, and according to the labeling team, the ratio of finding whales in the source imagery is approximately 7 whales in every 2000 images they manually checked, to create initial dataset.

Another challenge is the class imbalance in the dataset itself. In the dataset, not all of the classes have the same amount of examples. Approximately 85 percent of data hold three classes, and the remaining 15 percent belong to 16 other classes. Since machine learning models do not perform well on data with imbalanced classes, this challenge adds more value to the data collection downstream task.

The third challenge is the input image size. The dataset images' pixel resolution is high, which is 4912×7360 pixels in height and width, respectively, compared to the common benchmark image datasets (Table 1.1), which state-of-the-art computer

vision models usually are being developed using them. Subdividing these images into smaller images further exacerbates the problem of class imbalance, as most of the image is usually just water, and must be done so as not to accidentally lose a rare animal by having it present only partially on the boundary of sub-images.

Table 1.1: Examples of the generally used datasets for the object detection task with the number of their annotated images/objects in the training and validation set

	Number of Categories	**Spatial Resolution**	**Training**		**Validation**	
			Images	Objects	Images	Objects
DOTA-v1.0	15	800x800 - 20,000x20,000	1,403	94,141	468	31,380
PASCAL VOC	20	469x387 (on average)	5,717	13,609	5,823	13,841
ILSVRC 2014 (2017)	200	482x415 (on average)	456,567	478,897	20,121	55,502
COCO	80	$\sim$ 640x480	118,287	860,001	5,000	36,781
Stuff COCO	91	$\sim$ 640x480	118,287	747,458	5,000	32,801
Open Images V6+	600	$\sim$ 1024x1024	1,743,042	14,610,229	37,306	303,980

1.4 Overview

As mentioned previously , the objective of this project was to develop a computer vision model that can detect and classify marine animals in aerial imagery to help biologists find whales.

We startedby building an object detection model. We chose the Faster R-CNN architecture (discussed in Chapter 6) that has higher accuracy rather than one-stage object detection architectures. It is slower than one-stage architectures, but since our data is not video footage, inference speed was not our concern.

Our limited amount of data (discussed in Chapter 3) was insufficient to train the model, so we used transfer learning by fine-tuning a Faster R-CNN model trained on the COCO 2017 dataset. We used data augmentations (discussed in Section 5.3) to increase the number of samples as well.

We then needed to deal appropriately with the spatial resolution of the imagery (7360×4912), as it was significantly higher than the common imagery resolution that Faster R-CNN can handle (1200×800, keeping the original image's aspect ratio).

Downsampling was not feasible since most animals in the images are very small; hence, we needed to carefully crop images into smaller sub-images making sure to appropriately preserve animals in their entirety. (discussed in Sections 5.1, 5.2).

An ideal detection model would make no false negative predictions and few false positive predictions so that it is directly usable for monitoring whales. However, if that is not possible, another useful model would be one that makes no false negatives (i.e. we do not want to miss a whale that is present), but that makes *few enough* false positives that it is still helpful to an expert annotator in that it narrows down the options and *speeds up the annotation process* so that a larger dataset could be collected more efficiently, to eventually create a fully functional model. That is, even bootstrapping the data collection itself would be helpful. This would allow gradually building a set of increasingly accurate models, each of which in turn helps build larger data sets more efficiently and so on.

The number of false positive predictions in our object detection approach was high. Furthermore, high confidence values (> 0.5 and sometimes > 0.9) made it impossible to use different threshold values to eliminate them without also creating too many false negatives. The high false positive rate made it hard to label more data as the human expert should verify all the images.

We therefore next decided to try semantic segmentation approach to mitigate this problem. The idea is that since the labels are pixel-level for semantic segmentation, the model could see an ample amount of "empty" water (background) pixels, during the training, that could possibly lead to having fewer false positive predictions than object detection approach.

The proper annotation for semantic segmentation is polygon labeling. We first had the labeling team try polygon labeling. The time needed for polygon labeling was substantially higher than bounding boxes, so polygon labeling turned out to be infeasible. We therefore introduced elliptic labeling (discussed in Chapter 4) as a method by which to have annotations better than bounding boxes for semantic segmentation, while staying within the time budget of the labeling team. From the top view of aerial imagery, the objects have oval shapes, so they can be covered with elliptic masks with less amount of noise rather than covering them with rectangular masks, derived from bounding box labeling. These ellipses are also convertible to the bounding box labels,

so it does not prevent us from training object detection models using them. At the end of this phase, we could get elliptic labels for a portion of the dataset objects ($\approx$ 22%), and we used a mixture of elliptic masks and rectangular masks to train semantic segmentation model. We were aware that both the rectangular masks and elliptic masks are noisy annotations relative to what semantic segmentation usually requires.That is, both rectangular and elliptic annotations do include water pixels (background) incorrectly labeled as an animal. The central rationale behind using the noisy annotations was that since the number of background pixels is substantially higher than the objects, this could outweigh the noise, and the prediction masks could get shapes more proportional to the animals, i.e. neither perfect ellipses nor bounding boxes. This was successful; we did indeed observe this behavior to some extent in our qualitative results of semantic segmentation approach.

We chose DeepLabv3 as a state-of-the-art architecture (discussed in Chapter 7) with reliable implementation as the backbone architecture of our semantic segmentation approach.We fine-tuned a pre-trained (on `COCO 2017` dataset) model similar to the object detection approach, applying cropping and data augmentations. For this network, the proper crop size of the data is (600×400).

In parallel with development of the semantic segmentation approach, we also developed a technique for visualizing our dataset in large groups of globally similar images. This technique consisted of a grid containing images clustered using K-means based on features extracted using a ResNet architecture (discussed in Section 8.3.1). The purpose of creating this visualization was to get a better understanding of the dataset and the task and the scope of challenges we might be facing (e.g. what fraction of the images have significant glare from the sun on the water, making it extremely hard to detect and identify animals for both the labeler and computer vision model?). Interestingly, comparing the environmental annotations (discussed in Section 4.4) with the clusters validated our feature-based clustering approach (discussed in Section 8.3.4), and may suggest more sophisticated techniques that could be used in future that incorporate environmental conditions.

Having both semantic segmentation and object detection approaches, and motivated by the idea of developing a mechanism which would support a bootstrapping process for the collection of a large dataset, we finally compared them in a systematic

experiment (discussed in Section 8.4). This AI-in-the-loop experiment aims to find the best approach to reduce the amount of time needed to label more data and find a qualitative relation between numerical metrics and the time of labeling. The experiment has three conditions: manually labeling, labeling with the help of an object detection model, and labeling with the help of a semantic segmentation model. For the purpose of labeling, the masks are less important than the regions themselves. Therefore, to have comparable results, we converted the segmentation masks to the bounding boxes.

AI-in-the-loop adds complexity to the experiment that it would not be problematic in an AI-only context. Using the same data three times during the experiment for comparing three approaches would not produce fair results. The labeler sees them several times, and the result would favor the last condition that is being timed. On the contrary, as an example in an AI-only setup, the same validation set can be used several times in model selection. To overcome this challenge and conduct a fair comparison, we used K-means clustering and visual feature extraction to prepare data for training and analysis in this experiment. Doing so, we made sure that each condition has visually similar data but not the same as the other conditions for comparison (discussed in Section 8.3).

1.5 Contributions and approaches

In summary, this work includes the following contributions:

- We implemented two computer vision models based on Faster R-CNN and DeepLabv3 for object detection and semantic segmentation approaches to detect animals in aerial imagery.

- We leveraged the object-background class imbalance problem to allow use of noisy labels for semantic segmentation.

- We introduced elliptic labels for object annotation in segmentation to reduce noise in the ground truth masks while still maintaining efficiency for the expert human annotators.

- We designed and conducted an AI-in-the-loop experiment to show the influence of object detection and semantic segmentation models on data labeling compared to manually labeling images.
- We developed a clustered visualization of the dataset based on their extracted features, and qualitatively validated this for effective data preparation for the experiment, based on expert human annotators' workflow and environmental labels.
- We propose a bootstrapping approach for iterative dataset collection and model training in data-scarce contexts, and our results suggests that with even slightly more data than what was available to us, this would be feasible.

1.6 Thesis organization

We arranged the rest of the thesis as follows: In Chapter 2, we briefly discuss the related research to this work. After describing the data in Chapter 3 on the dataset, then we introduce the methods used to label and annotate the data in Chapter 4. Chapter 5 is on Pre-processing and how the data is being processed to be suitable for the architectures we used. Chapter 6 and 7 are on the object detection (Faster R-CNN) and semantic segmentation (DeepLabv3) models, respectively. In Chapter 8, we discuss the approaches we used to train and evaluate our object detection and semantic segmentation models. Finally, in Chapter 9, we have the conclusion and the future works.

Chapter 2

Related Works

In this chapter, we briefly discuss some of the related works to this research. We grouped related works into three broad categories: object detection, semantic segmentation, and works at the intersection of biology and computer science for solving animal detection and classification in the water.

2.1 Object Detection

Object detection is a computer vision task that, given an image, predicts the location of the objects present in images as well as their class. The evolution of this field can be divided into two eras of before and after the wide usage of convolutional neural networks (CNNs). Before CNNs, traditional object detections such as Viola-Jones Detectors [12,13] and HOG detectors [14] were standard algorithms using hand-crafted features. HOG detectors evolved into Deformable Part Models (DPMs), and DPMs [15–18] were the winner of the PASCAL VOC for three consecutive years of 2007 to 2009. DPMs are graphical models in essence, but it was shown by Girshick et al [19]. that DPMs are also can be formulated as CNNs.

AlexNet [20] heavily influenced the field of computer vision in general, and object detection was one of the affected topics. AlexNet utilized CNNs, which had already been used earlier in image classification networks such as LeNet-5 [21]. The detection pipelines that utilize CNNs are often categorized into **one-stage** and **two-stage** detectors. In two-stage detectors, there is a region proposal mechanism prior to the primary detection. Figure 2.1 illustrates these two categories with their corresponding timestamps of development.

2.1.1 Two-Stage Detectors

R-CNN: Regions with CNN features [22] was one of the earliest steps in the field of object detection with convolutional neural networks. This two-stage object detector

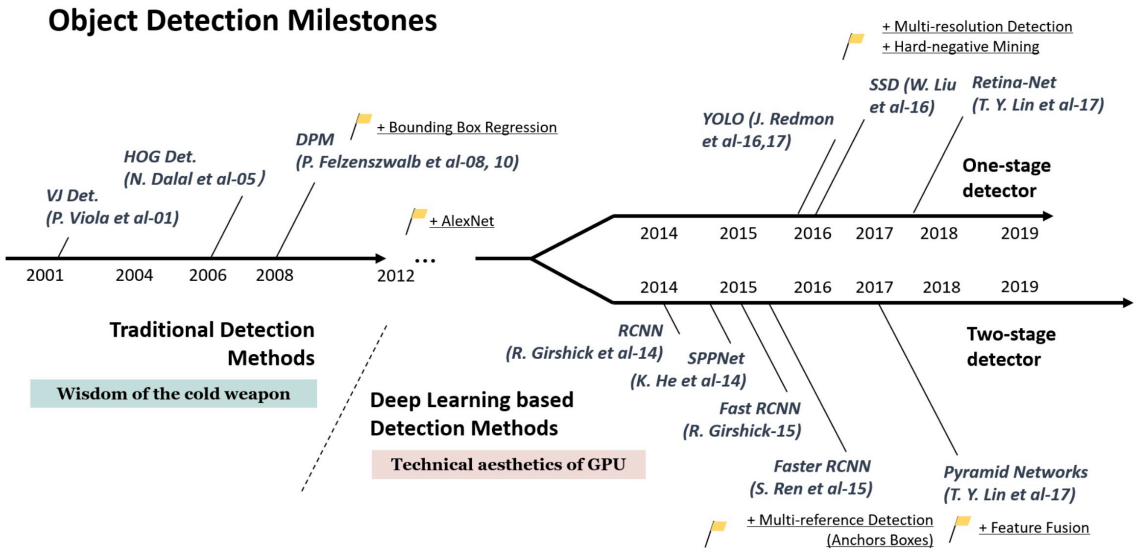


Figure 2.1: The object detection algorithms' milestones. The image is taken from [1]

had a significant speed problem that was not suitable for real-time usage. Soon after R-CNN, another two-stage object detector, Spatial Pyramid Pooling Networks (SPPNet) [23], was proposed to overcome the challenge of speed. The main drawback of this architecture was that only the fully connected part of the network was being fine-tuned on the data. In addition to that, the training process was multi-stage. Fast R-CNN [24] had the advantage of fine-tuning both detector and the fully connected bounding box regressor parts of the network. Fast R-CNN was not suitable for real-time detection, and Faster R-CNN [5], the latest version of R-CNN-based detection pipelines, was proposed that was an end-to-end trainable deep network. One of the critical challenges in object detection was the presence of multi-scale objects. Feature Pyramid Networks (FPN) [4] was proposed to make the Faster-RCNN more capable of detecting objects presented in multiple scales and one of the reliable and accurate object detection pipelines. We chose this pipeline for the object detection approach that is described in Chapter 6.

2.1.2 One-Stage Detectors

In one-stage detectors, You Look Only Once (YOLO) [25] was the first research that later on went through a series of improvements to shape v2 [26] and v3 [27] versions of it. The authors of the first work proposed the two later versions. Later in 2020, v4 [28]

was proposed by another set of authors. In the YOLO algorithm, the idea is to slice the input image into a grid and feed the image into a single CNN to get the feature map and compute the probability of the presence of an object in any of the cells in the grid. Despite the higher speed of prediction in YOLO, the error rate of YOLO in bounding box prediction was initially high [1]. Improving localization accuracy was one of the goals in developing the newer versions of YOLO and Single Shot MultiBox Detector (SSD), which was proposed by Liu et al. [29]. SSD had a performance with mAP of 74.9% on VOC 2012 comparing to YOLO with mAP=57.9%. Eventually, Tsung-Yi Lin et al. proposed RetinaNet [30] utilizing focal loss, making it one of the best one-stage detectors. The focal loss was a customized cross-entropy loss with a lower weight to well-classified cases.

The research in object detection is not limited to the one and two-stage object detectors. With the advent of transformers and their superior performance in sequence modeling, object detection task also has been influenced by them. End-to-End Object Detection with Transformers (also known as DETR) [31] is one of the recent frameworks that utilize transformers to predict bounding boxes for the objects present in the images. They used transformer architecture as a replacement for the object detection pipeline in Faster R-CNN architecture. Due to its slow convergence, a newer transformer-based architecture was proposed, namely deformable DETR [32]. PIX2SEQ [33] is another recent approach by Google Brain that defines object detection as a language modeling task that reaches a competitive result comparing to the other high-performing object detectors.

2.2 Image Segmentation

Image segmentation is a task in computer vision that divides the input image into regions that share the same semantics. Image segmentation has three main sub-tasks: **instance segmentation**, **semantic segmentation**, and **panoptic segmentation** [34]. In instance segmentation, the goal is to predict objects, or in other words, the instances of the classes of the objects. Unlike object detection, instance segmentation predictions have pixel-level labels for each instance. Inspired by the success of Faster R-CNN architecture, Mask R-CNN was developed, with a similar

architecture to Faster R-CNN. Mask R-CNN is a successful instance segmentation architecture proposed in 2017 [35]. Semantic segmentation predicts pixel-level labels for every possible pixel in the input image without considering the objects individually. For example, in the case of having two overlapping objects of a class, the output prediction assigns the same value for all of the pixels covering these two objects without making any distinction between those. Unifying the two tasks of instance and semantic segmentation, the panoptic segmentation, predicts two values for each pixel in the input image: class id and object id. Proposed by Kirilov et al. [36], panoptic segmentation task became one of the computer vision research tracks. Panoptic-DeepLab [37] can be named as one of the best algorithms in the panoptic segmentation task. In this section, our focus is on semantic segmentation task.

Semantic segmentation task was tackled with graphical models and clustering algorithms before deep learning, but similar to other vision tasks, deep learning heavily affected semantic segmentation. Fully convolutional networks (FCN) [38] was one of the earliest attempts to solve semantic segmentation. In FCN, the network is a modified classification network, such as AlexNet, VGG [39], or GoogLeNet [40]. They took the classification network and substituted the last fully connected layers with convolutional layers, making the network fully convolutional. In FCN, they combined outputs from several stages to produce the final prediction.

Encoder-Decoder design pattern in deep learning has been a blueprint for many segmentation architectures. DeConvNet [41] is one of the Encoder-Encoder style segmentation architectures that uses convolutional layers in the encoder sub-network that downsamples the feature map and deconvolution layers in the decoder sub-network to recover the spatial resolution. SegNet [42] was another approach that adapted VGG for the encoder, but for the decoder, they used the unpooling method in which the indices of max-pooling operation in the encoder sub-network were being stored to be used for the upsampling purpose in the decoder sub-network. U-Net [43] is one of the significant architecture in this category. It was proposed by Ronneberger et al. that was initially developed for biomedical image segmentation. U-Net is has a symmetric architecture that activations in each step in “contracting path“ (the encoder) are concatenated to their correspondings in “expansive path“ (the decoder).

Dilated convolution (also known as atrous convolution), proposed by Holschneider

et al. [44], is the building block of many successful works in semantic segmentation. Dilated convolution is a type of convolution operation such that several zeros are introduced in the convolutional kernel between the values, producing bigger filters without changing the number of learnable parameters. For example, a dilated variant of a 2×2 convolutional filter could be a 3×3 filter with the same values of the original 2×2 filter spread out into the four corners of a 3×3 filter, while the rest of the elements are zero. Receptive field can be controlled explicitly using dilated convolution without stacking several convolutional and pooling layers. We described this operation in detail in Section 7.2.1. The successful DeepLab family of networks is using this convolutional operator to leverage more context in classifying input image pixels. DeepLabv1 [45] and DeepLabv2 [8] are incorporating conditional random fields for the refinements in the prediction, while DeepLabv3 [46] and DeepLabv3+ [47] are discontinued using CRFs, making them fully convolutional architectures. In this project, we used DeepLabv3, which reached 85.7 on the mIoU score on the PASCAL VOC 2012, for the semantic segmentation approach. We describe this architecture in Chapter 7 on semantic segmentation

2.3 Detecting Whales

Detecting and surveying whales using acoustic data has always been an ongoing effort. Several research works have been done to facilitate this data domain to locate and classify cetaceans in the water [48–56], including machine learning and deep learning methods [57–67]. However, this type of data has an intrinsic uncertainty of localization that deteriorates the accuracy of locating marine animals [68].

Aerial imagery is another domain of data to be utilized for surveying and detecting cetaceans. There are a few attempts to automate surveying whales using aerial imagery and high-resolution images captured from space from decades ago [69]. Several recent studies have been conducted to study the feasibility and usefulness of space imagery in detecting and classifying cetaceans [70–73]. Guirado et al. had a study [74] on counting whales in satellite and aerial imagery. They proposed a two-step framework with consecutive classification and object detection models. They fine-tuned the classification network, pre-trained on ImageNet, on their dataset containing 2100

images evenly distributed into three classes. For the object detection phase, they fine-tuned only the last two layers of Faster R-CNN, pre-trained on COCO dataset. The image classifier network gets the input image in a sliding window fashion to detect the tiles containing whales. Then tiles are being fed to an object detection model for counting the whales present in them. Their approach reached 81% and 94% of F-1 measure on their proprietary dataset for image classifier and object detection model, respectively. However, the study has two technical drawbacks. They did not fine-tune all the network weights for object detection architecture, and their framework is not an end-to-end differentiable architecture for detecting whales. Borowicz et al. proposed an approach of training an image classifier on downscaled aerial imagery to survey cetaceans from satellite imagery [75]. Their approach was based on tiling. They used aerial imagery that contained 190 images of 17 minke whales to train image classification networks, ResNets and DenseNet. For evaluation, they used satellite imagery in which the challenging images were excluded. Utilizing this domain can be beneficial for human activities as well when the animals are dangerous for people, such as shark detection. Sharma et al. conducted a study of detecting sharks in the vicinity of beaches using aerial imagery with the help of Faster R-CNN model to alert about the presence of sharks for people's safety [76].

Software platforms is another track that researchers are developing for the purpose of conservation. Wildbook [77] is an open-source platform for crowdsourcing the effort needed to conserve endangered species and not only whales. WhaleMap [78] is a software platform to collect and display survey data. The platform collects surveys from different sources such as planes and vessels and displays them in near real-time.

Chapter 3

Dataset

In this chapter, we will describe the dataset we have had during this work.

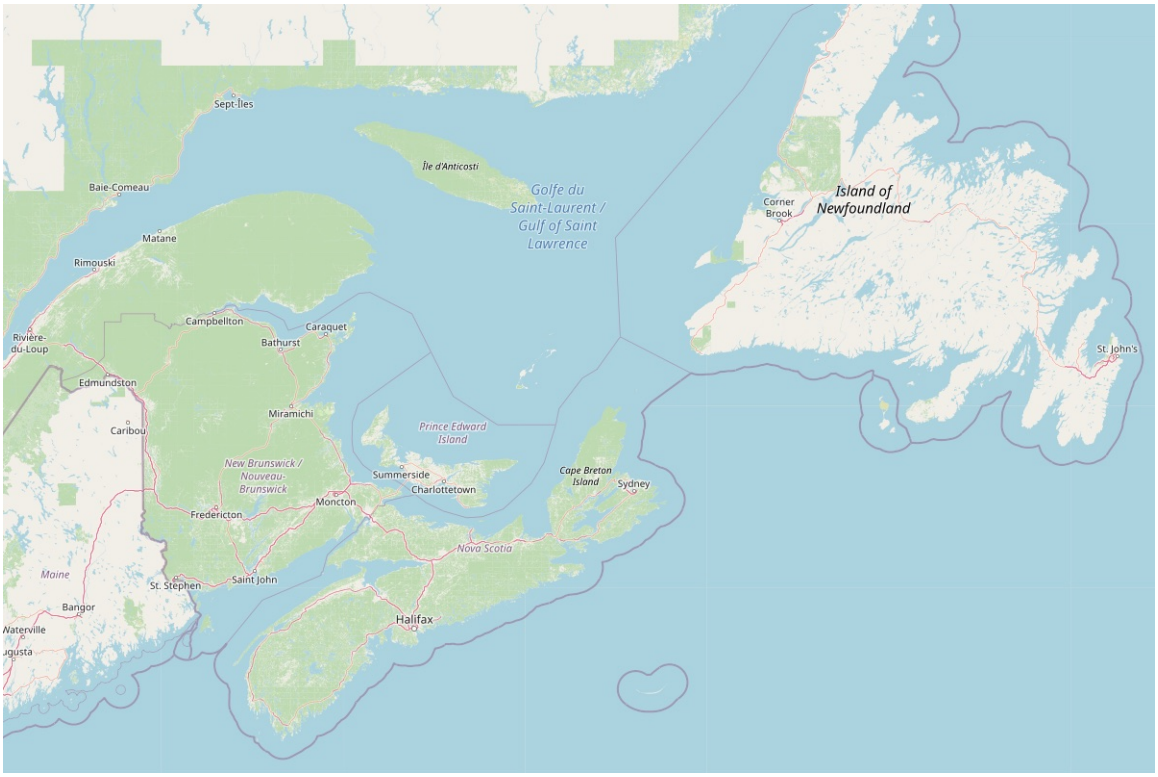


Figure 3.1: Gulf of Saint Lawrence. The map is exported from **OpenStreetMap** [2]

3.1 Image List

Generally, the dataset description images and samples are provided in the corresponding sections. We provide images in sections that can benefit the most from them for the purpose of clarity. But for the convenience of the reader who wants to see the images before reading about them, here is the list of images: Figure 3.2, Figure 3.3, Figure 3.4, Figure 4.6, and Figure 8.3.

3.2 Data Acquisition

The images are captured and gathered by **DFO** using in a **nadir** pointing setup of two (left and right) cameras mounted on an aircraft. These two cameras were constantly shooting images over the Gulf of Saint Lawrence region, which is depicted in figure 3.1. Each image may or may not contain one or more animals. The majority of images are aerial imagery containing only water and occasional shoreline.

3.3 Data Descriptions

The data samples are images depicting an area from the overhead. The color mode of these images is RGB, and they are stored in JPEG format. The spatial resolution of all images is 7360×4920 pixels. Figure 3.2 shows a sample image. We have had 1544 images to develop the models based on them, all of which include at least one example of the objects discussed in the next section.

The mean object to background ratio among all images is 0.16 percent (calculations are available in Appendix A), making it challenging to find and identify the objects without zoom and panning for experts to find the objects manually and for algorithms to detect objects. Figure 3.3 illustrates a turtle presented in an image (the image is the same image in figure 3.2) with a $20\times$ zoom level

Due to the various weather conditions and lighting at the time of capturing, the amount of noise and colors vary. Figure 3.4 shows several sample images containing objects. Objects are not visible in that scale due to the low ratio of the foreground (objects of interest) to the background.

3.4 Labels

Two major categories of annotations have been collected for this dataset: object annotations and environmental tagging. We discussed each type of annotation and the procedure of obtaining them in the Chapter 4 on labeling.

Several images have more than one object totaling 4892 labels in the 1544 images. The pie chart in Figure 3.5 shows the proportions of each label. Fishing gear comprises the majority of the dataset with 2198 samples, and Humpback whale with one sample is the rarest class of objects.

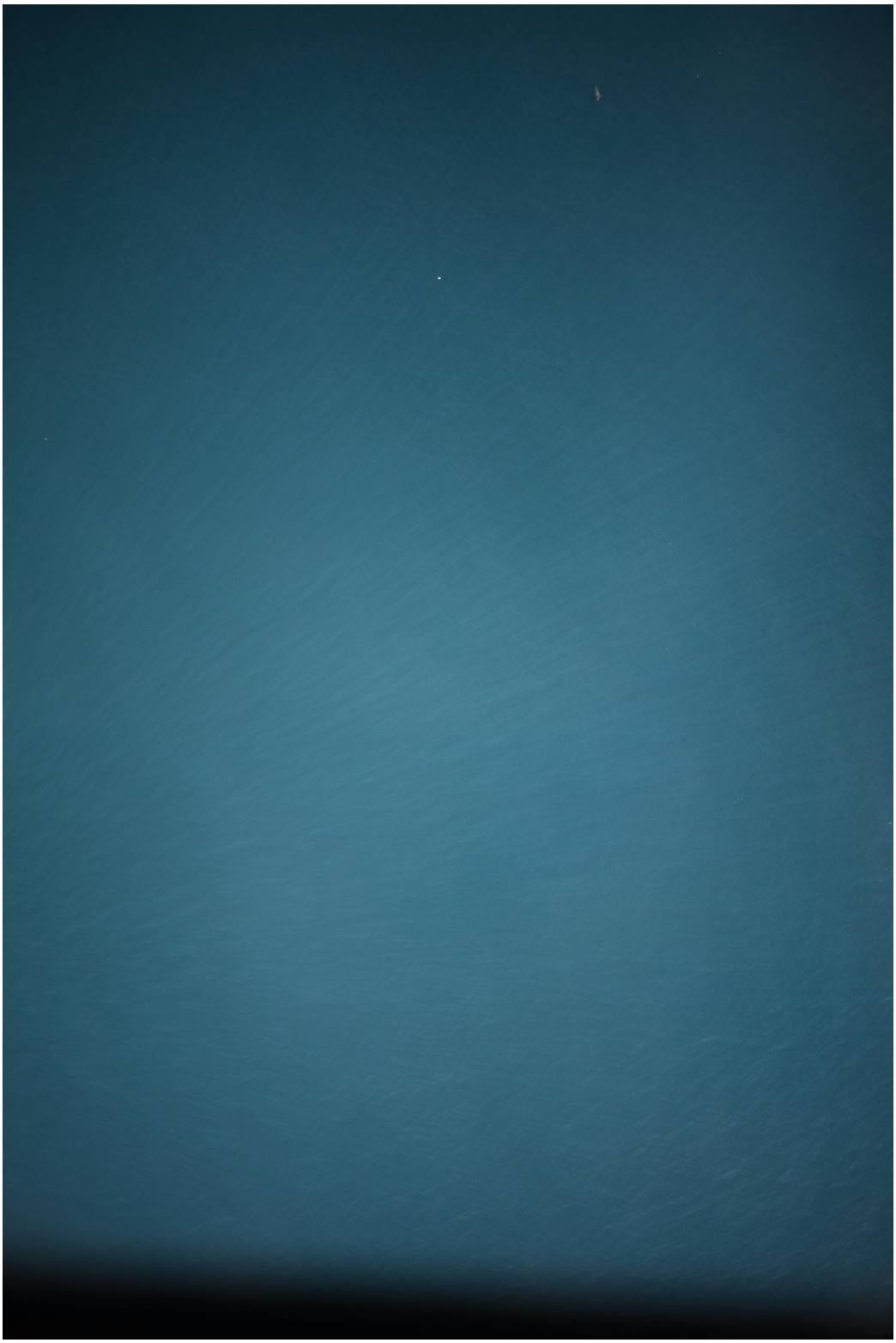


Figure 3.2: A counter-clockwise 90°rotated example image.

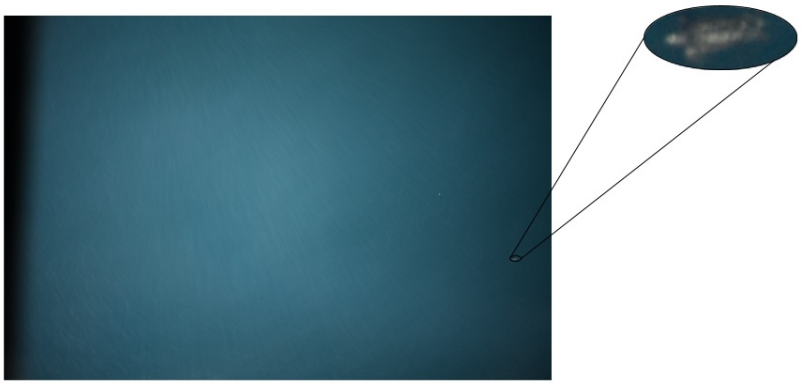


Figure 3.3: An example image with an object visible at 20× zoom level

While some of the labels are clear from their names, others need descriptions around them. `Animal` is a class of objects that are difficult to recognize due to their small sizes or other unfavorable conditions that hide their significant characteristics. The `Object` class contains any distinguishable solid pattern, such as debris. The `Artifact` is a class of objects that are not solid entities but instead temporary illusionary patterns such as waves or fractured patterns caused by animal movements or sudden changes in the water. There are also several classes with abbreviated names. `WNR` stands for `Whale Not Right`, referring to whales that are not right whales. `LBWNR` and `ULA` are two other abbreviations standing for `Large Baleen Whale Not Right`, and `Unidentified Large Animal`, respectively. `ULA` is a class of animals that are large enough to be distinguished from animal class but without the details needed to be precisely recognized. Finally, `Right` is the short form for `Right Whale`. Table 3.1 has all of the classes with their summarized descriptions and number of samples.

A part of our industrial collaboration involves justification and the rationale behind having the object labels other than animals and cetaceans. Imagine a self-driving car that needed to detect and recognize green lights and red lights. Imagine there were no trees in the training data of it. So, this car will learn that it is okay to go whenever it sees green in the sky, which is not acceptable. Suppose we train our object detection and semantic segmentation models without fishing gear, boat, and fishing line examples. In that case, the models become a detector for those since the model learns that whatever is not water is an animal and should be detected.

Table 3.1: Object classes and their description and number of samples in the dataset

Class Name	**Summarized Description**	**# of Samples**
`Fishing Gear`	Fishing Gear	2198
`Animal`	Animals that are impossible to identify due to their small sizes or other factors that adversely affect the recognition procedure	1652
`Object`	Solid patterns that are not animal such as debris	297
`Artifact`	Structures and patterns are not objects created randomly like waves that look like objects, but a fractured surface implies nothing but water or wave.	165
Fishing Line	Fishing Line	149
`Boat`	Boat	121
`Minke`	Minke Whale	97
`Sunfish`	Sunfish	82
`WNR`	Whale Not Right: Whales that are not Right Whales	24
`Shark`	Shark	22
`Right`	Right Whale	18
`Basking`	Basking Shark	16
`LWNR`	Large Baleen Whale Not Right: Other species of Baleen Whales but not Right Whale	11
`Whale`	Whale	10
`ULA`	Unidentified Large Animal: Large animals, but not accurately recognizable	8
`Fin`	Fin Whale	7
`Blue Shark`	Blue Shark	7
`Turtle`	Turtle	6
`Humpback`	Humpback Whale	1

Table 3.2: The set of environmental tags and their description

Environmental Tag	Description
`High/Medium/Low Glare`	The amount of glare presented in the image
`Rough/Calm Seas`	The state of water
`Debris`	Presence of debris or similar objects
`Land`	Presence of land in the image
`Cloudy`	Considerable amount of cloud
`Night`	The image was taken in night
`Unknown`	There is no tag provided for the image

The other type of annotation we gathered and utilized is environmental tagging. We used these tags in data preparation. Environmental tags can provide information about the general appearance of images based on three criteria: sea state, glare amount, and special tags. Environmental tags comprise of `High glare`, `Medium glare`, `Low glare`, `Calm seas`, `Rough seas`, `Debris`, `Land`, `Cloudy`, `Night`, and `Unknown`. Land represents those images that contain any parts of the land, such as shorelines. The `Unknown` tag represents the images that do not have environmental tags, and the rest are self-explanatory. This scheme was not a comprehensive approach, and not all samples have tags for all of the three aforementioned criteria. Table 3.2 has a summary of these descriptions for the tags. Figure 3.6 is the distribution of environmental tags. Forty-three images do not have any tags and are categorized under the `Unknown` label, and the rest of the tags are shared between 1501 images.

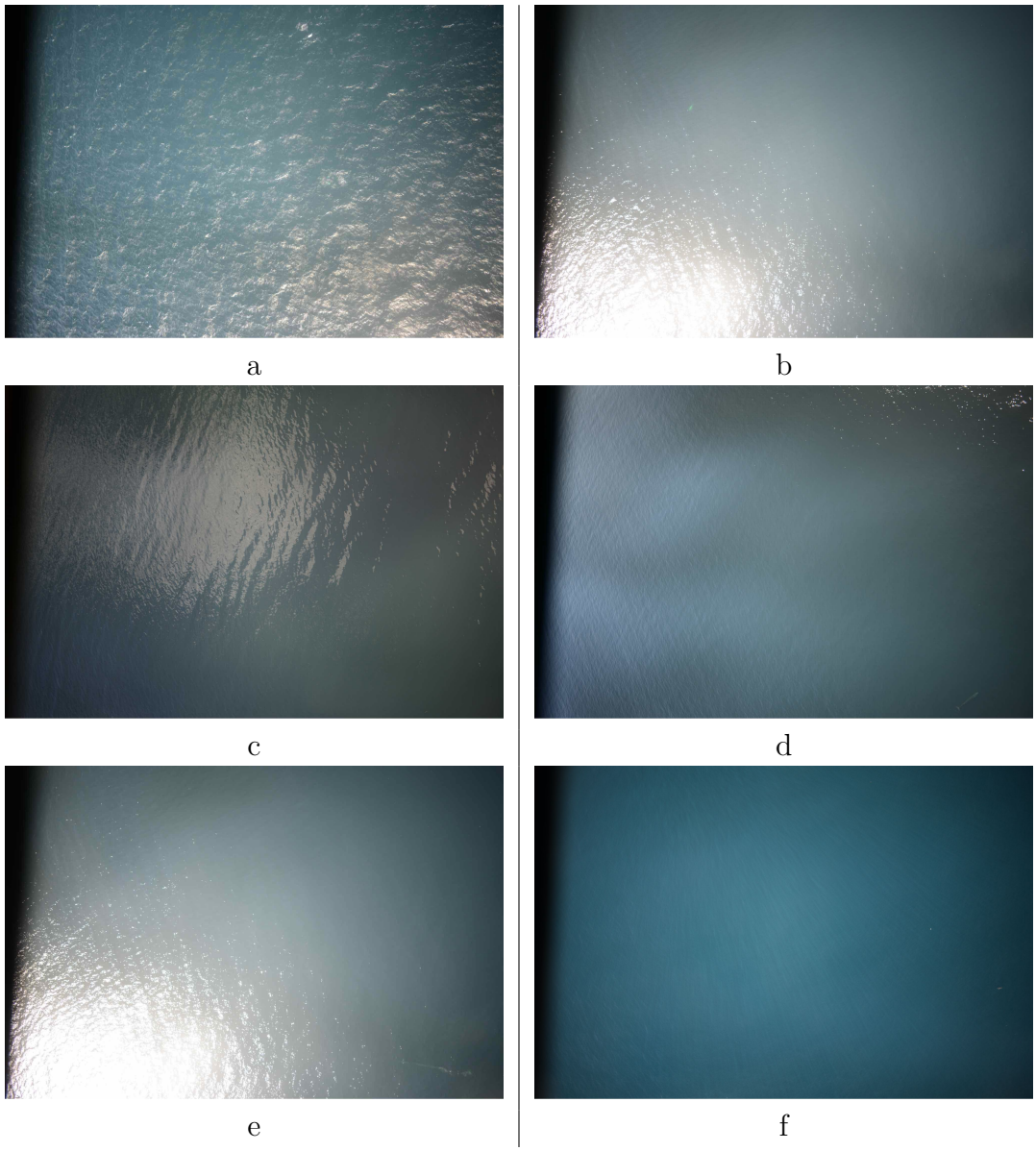


Figure 3.4: Several sample images from dataset:
a: The image contains a Humpback whale
b: The image contains an Unidentified Large Animal and an Object (Not animal)
c: The image contains no animal, but some patterns are there due to the harsh environment
d: The image contains a Large Baleen Whale Not Right
e: The image contains a Minke Whale
f: The image contains a Turtle

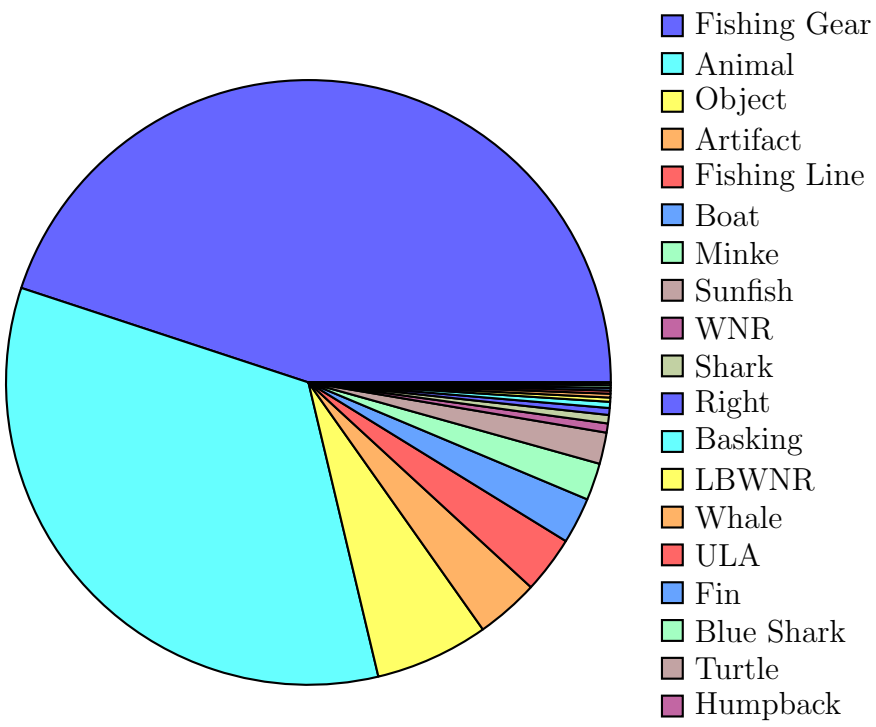


Figure 3.5: The pie chart of the dataset

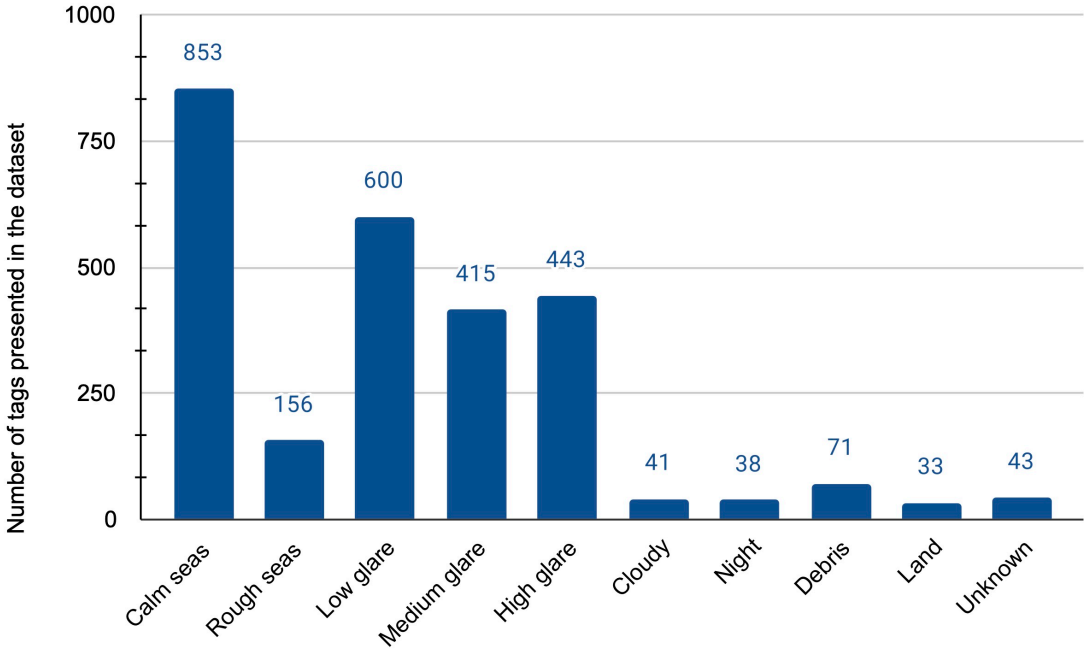


Figure 3.6: Distribution of Environmental Tags

Chapter 4

Labeling

In supervised learning, labeling has significant importance. Annotating images for training deep neural networks was an ongoing part of this project, and we learned as we went forward. We changed the schemes based on our needs. We tried object detection approach, and for that, we needed bounding box labeling. Then we wanted to try semantic segmentation out. Then we wanted to try semantic segmentation, and semantic segmentation needs polygon labeling. Polygon labeling was not a feasible option given the limited time resources available. We introduced a new way of labeling that is more accurate than the bounding box but less costly than polygon labeling. A team of biology researchers was the experts who had done the annotating of objects. In this chapter, we discuss those labeling procedures.

4.1 Bounding box labeling

In the typical object detection task in computer vision, the model predicts tight axis-aligned bounding boxes around the objects, so the original way of annotating images for this task was to annotate objects with boxes similar to those that the would model predict. In this labeling scheme, each object of interest is annotated with an ordered set of four points, defining a tight bounding box containing it with a class label.

The bounding box labeling requires a rectangle for each object, and a rectangle can be represented with its diagonal and needs at least two points for this representation. Each point in a 2-dimensional (2D) plane needs two values for its x and y coordinates. Therefore, the minimum number of values for representing a rectangle in a 2D plane is four.

There are two common ways to collect the bounding box labeling. One way is as a 4-tuple (x_0, y_0, h, w), where the first two scalars correspond to the top-left corner, and the other two represent the height and width of the axis-aligned box. **COCO**, one of the benchmark datasets in object detection, uses this way of labeling bounding boxes.

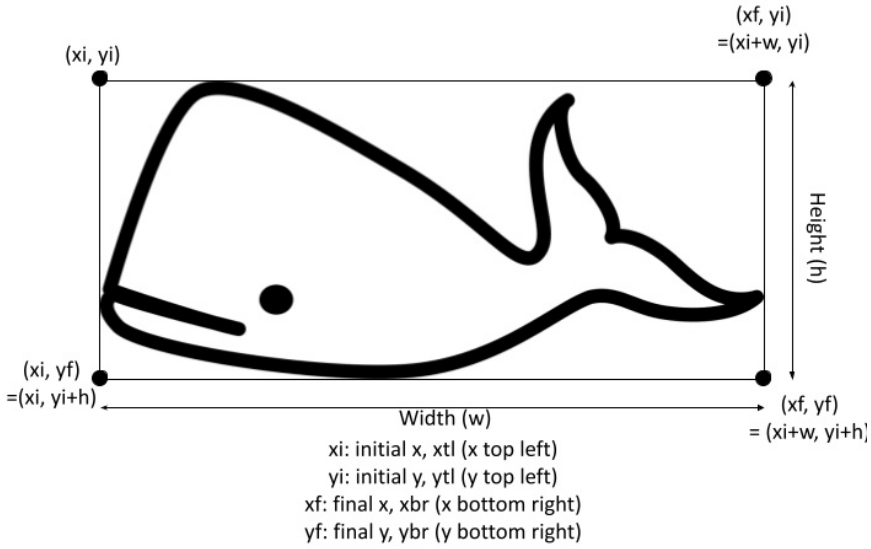


Figure 4.1: Bounding Box Labeling Schemes

Another approach is capturing four values of x and y coordinates of the diagonal of the rectangle directly. In this project, similar to the **PASCAL VOC** dataset, we used a 4-tuple $(x_{tl}, y_{tl}, x_{br}, y_{br})$ for defining the bounding boxes around the objects, which **tl** and **br** are referring to top-left and bottom-right, respectively. Figure 4.1, depicts the relationship between these points and the objects they are covering.

4.2 Polygon labeling

Axis-aligned bounding box labeling is a simple way to annotate objects. However, this scheme is not suitable for semantic segmentation since the predictions are made at the pixel-level in that task. Therefore, the labels should have pixel-level information about the data. Using bounding box labeling for semantic segmentation makes the data annotations excessively noisy, which not only deteriorates the learner's performance but also makes the metrics inaccurate. In this chapter, we use the term "noise" to refer to the incorrectly labeled pixel(s). For example, Figure 4.2 contains an animal in a bounding box; using the bounding box as the pixel-level label of the object yields the gray area around the animal to be marked as the pixels containing that animal, which is incorrect.

For this reason, polygon labeling is more proper for the pixel-level predictions of the semantic segmentation task. In polygon labeling, the goal is to find the exact

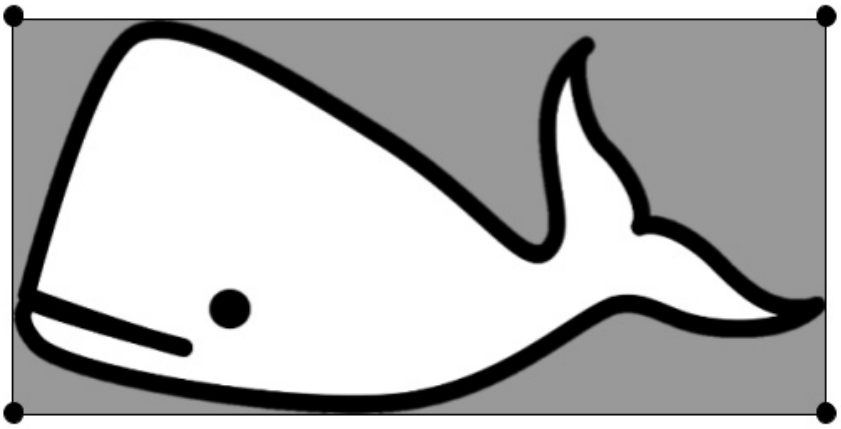


Figure 4.2: Despite being a flawless bounding box for object detection, this annotation has a noticeable amount of noise for semantic segmentation.

shape of the objects present in the image. The exact shapes of objects contain pixel-level information that is suitable for semantic segmentation.

In polygon labeling, the labels are a set of edges in (x, y) coordinates defining the polygon's edges containing the object and the label corresponding to the object's class. Connecting the edges with straight lines yields the object's shape with crisp edges. This scheme includes only a tiny amount of noise produced by the labeling process and human error. Figure 4.3 depicts the polygon labeling with an intentional noise around the object for demonstration purposes. Polygon filling algorithms propagate the object's class to the pixels of the image inside the polygon to produce masks that are target values in the semantic segmentation task.

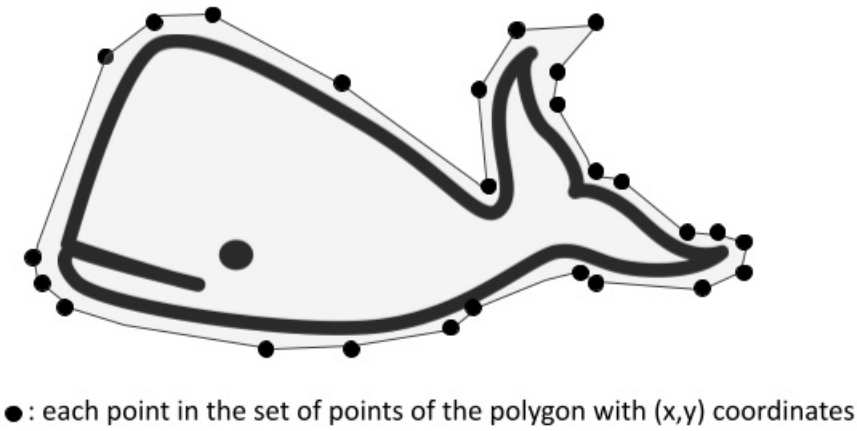


Figure 4.3: Polygon Labeling Schemes

Polygon labeling was not a feasible option for this project. Given the limited time the labeling team had, we could not use polygon labeling since it is time-consuming.

4.3 Elliptic Labeling

Although polygon labeling is the general way of image annotation in semantic segmentation task, it is highly time-consuming. Polygon labeling can be interpreted as the generalization of bounding box labeling where the number of points required to determine the polygon is higher than four. This is the source of extra time consumption in the polygon labeling in comparison to bounding box labeling. Hence this is not always feasible to have accurate labels given available resources.

We introduced a method called elliptic labeling that is less time-consuming than polygon labeling and less noisy than bounding box labeling. Objects in the imagery are captured from the top view and have elongated and oval shapes that make the elliptic curve a better curve than the rectangle to mark the objects' shapes. Therefore, elliptic labeling using the axes of the ellipse containing the object is a suitable choice for the data.

In a 2D plane, an ellipse can be represented by the lengths of major and minor axes, the rotation angle of the major axis from the positive horizontal axis, and the position of the ellipse's center. These four parameters are also computable using the four ends of the major and minor axes. Hence, an ellipse can be represented by two line segments corresponding to its axes.

In elliptic labeling, the expert draws a line on the object, which usually lies on the animals' spine as the major axis and another line perpendicular to the first one and from the middle point of it as the ellipse's minor axis from one end to another end of the object. The four points generated in this process turn into an ellipse around the object.

A significant benefit of using the elliptic labeling scheme is the amount of captured noise, especially in the semantic segmentation task. Based on the rotation angle of the ellipse, the difference in noise between two schemes of bounding boxes and elliptic labeling varies from the minimum when the angle is zero depicted in figure 4.4 and the maximum when the angle is $\pi/4$ shown in figure 4.5. The ellipses in the figures are approximately the same size, so the difference in the gray area of the bounding

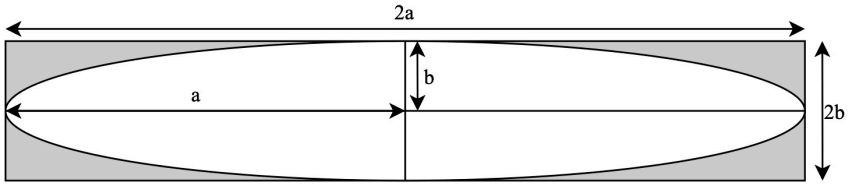


Figure 4.4: An ellipse with the rotation angle of 0. The gray colored area is the amount of extra noise that could possibly captured using bounding box labeling

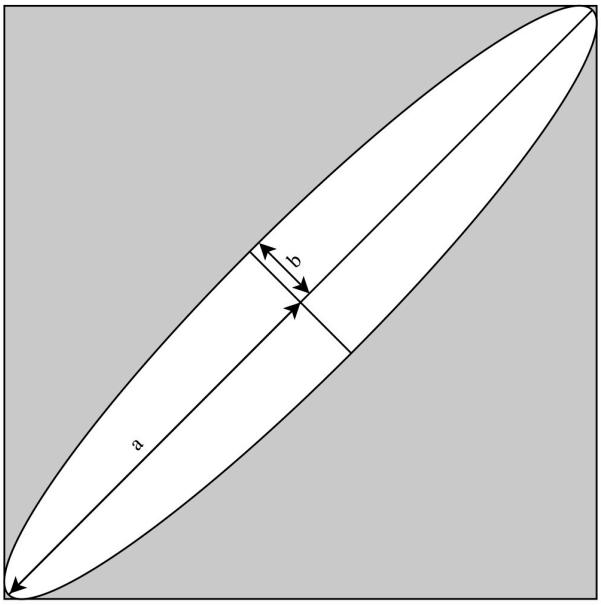


Figure 4.5: An ellipse with the rotation angle of $\pi/4$. The gray colored area is the amount of extra noise that could possibly captured using bounding box labeling

boxes is evident.

The number of free parameters that should be defined in elliptic labeling is double the amount of bounding box labeling needs. Elliptic labeling needs 4 points to be defined as the endpoints of the line segments, which are major and minor axes of the ellipse. Each point needs a 2-tuple (x, y), so each annotation in elliptic labeling needs eight parameters while bounding box labeling needs four parameters. In the dataset, 1104 objects have elliptic labels. The ellipses cover approximately 57 percent of the rectangles in those objects, giving more than 40 percent noise reduction.

Algorithm 1: Ellipse Extraction

Input: List of *Lines*

Output: List of *Ellipses*

1: $Intersecting\_Lines$ = Verify_Intersection($Lines$)
2: $Ellipses = \{\}$
3: **for** $(l_1, l_2) \in Intersecting\_Lines$ **do**
4: **if** Length($l1$) >Length(l2) **then**
5: $Major = l_1$
6: $Minor = l_2$
7: **else**
8: $Major = l_2$
9: $Minor = l_1$
10: **end if**
11: $\theta = \arctan \frac{Major_Y}{Major_X}$
12: $Center = (\frac{Major_X}{2}, \frac{Major_Y}{2})$
13: $Ellipse = (Center, \theta, (l_1, l_2))$
14: Add *Ellipse* To *Ellipses*
15: **end for**
16: **return** *Ellipses*

Since we introduced this method for this project, here we discuss the implementation details to elucidate the process. To implement this scheme, we used line annotation that was natively supported in the labeling software. In line annotation, the output contains four values for endpoints with a class associated with it for each line segment. Hence, in practice, annotations are line segments. Algorithm 1 contains the algorithm that converts lines into ellipses. Theoretically, real numbers have an infinite number of decimal point digits, making it practically impossible to divide them into two identical numbers. Reflecting this, it is invalid to expect the labeling expert to annotate the major and minor axes of the ellipse where they should intersect each other in the middle with an exact right angle. Still, it is possible to use this approach with an **assumption:** there should not be a line with two intersections within a

plane of lines. With this assumption, the ellipse extraction algorithm finds pairs of intersecting lines. `Verify_Intersection` (line 1 in Algorithm 1) makes sure to have one intersection per line segment to hold the assumption and returns a list of 2-tuples of intersecting line segments. Then, for each pair of intersecting lines, one becomes the ellipse's major axis based on the line segments' lengths, and the other becomes the ellipse's minor axis. We **assume** that lines intersect each other at the midpoint of the major axis so that the midpoint becomes the center of the ellipse. The angle of the ellipse is the arctan of the rise over run of the major axis. These parameters make it possible to fill the ellipse for semantic segmentation or find boundaries for object detection.

4.4 Environmental Tags

Environmental tags are a type of label we collected for experimental purposes. Ultimately, we have not used them in training models, and we adopted them for cluster verification purposes in data preparation (please refer to Section 8.3.4). The general (global) appearance of images can vary drastically between instances. In some images, the water is clear, and the lighting is suitable, while other images are noisy and dark. The labels for these conditions can help in devising an approach to train more robust machine learning models. Based on the condition of each image, environmental tags are grouped into three categories of **sea state**, **glare amount**, **conditional tags**.

This type of labeling and its procedure was not consistent throughout the project. The latest approach we decided to follow is to tag images based on three criteria of sea state, glare amount, and conditional tags. Sea state can be `Rough` or `Calm`, and glare amount can be one of three nominal values of `Low Glare`, `Medium Glare`, and `High Glare`. There are four other tags of `Debris`, `Land`, `Cloudy`, and `Night` are conditional tags. Sea state and glare amount tags are based on the general appearance of the images, but conditional tags are based on the presence of a certain factor or object in an image. Figure 4.6 contains 500×500 crops of images to demonstrate how each tag corresponds to certain characteristics of the images. Although calm seas and low glare look similar to each other, they are not the same. Sea state is an intrinsic characteristic and usually depends on waves, winds. However, glare is an extrinsic

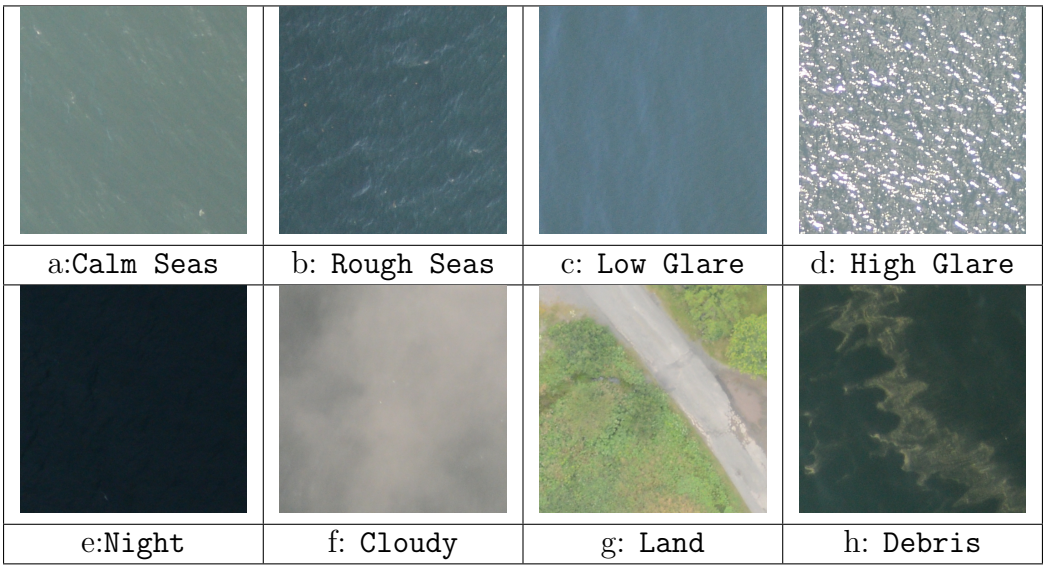


Figure 4.6: Examples of Environmental tags

factor to the water and has a connection with the time of image capturing, angle, and sunlight.

Chapter 5

Pre-processing

The common practice in computer vision tasks in deep learning contains minimal pre-processings such as subtracting the mean of the data from the training examples [20]. Since for some variations that are present in the data, it is beneficial for the model to learn and be aware of them [79].

For this research, the pre-processing can be divided into two groups of **online pre-processing** and **offline pre-processing**. The offline pre-processing contains only the input image size adjustments (cropping) to the appropriate sizes that models can handle. The online processings consist of normalization and data augmentations.

The dataset images' pixel resolution is 7360×4912 pixels in width and height, respectively, with three RGB channels. This amount of pixel is roughly 37 times bigger than the usual size of the images that deep object detection networks work well with, which is 1200×800, keeping the aspect ratio. The challenge is more difficult in the semantic segmentation task since the higher volume of computations needed for the semantic segmentation, the proper image size possible to fit in the memory of GPUs is even smaller.

One procedure of overcoming this challenge is to downsample the images into the proper size, but this approach is not applicable for this dataset. In aerial imagery, the ratio of object area to the background pixels is small. Therefore, downsampling images has the downside of losing objects of interest in the images. The majority of objects are packed in smaller sizes, and roughly 57% of the objects have less than 37 pixels in height. So downsampling ends up wiping objects. Hence, the better way of dealing with this problem is to cut the image into pieces and feed those patches into the model. In this research, two methods were used to crop the images into smaller patches: **grid cropping** and **random cropping**. The output images of these offline pre-processing methods, comprise the data used to train neural networks.

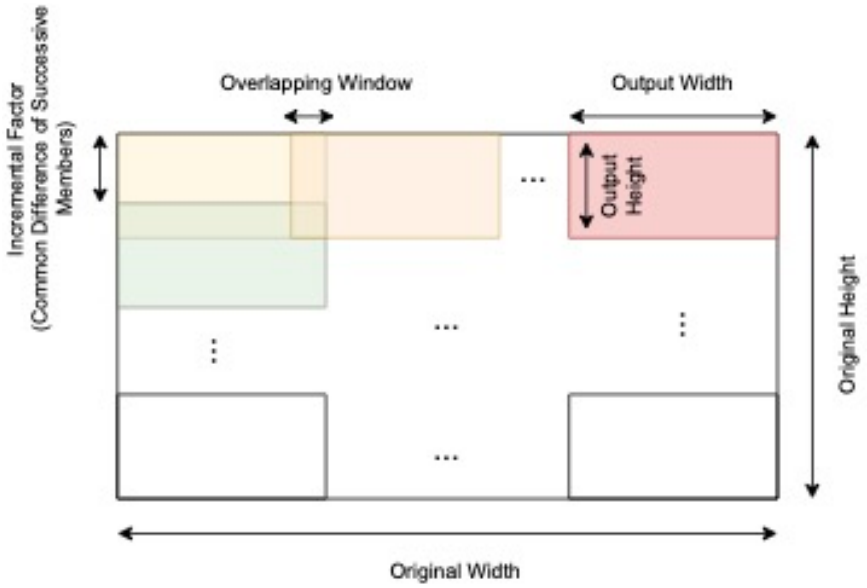


Figure 5.1: Grid Cropping Illustration

5.1 Grid Cropping

Cropping images into small patches can be done by interpreting images as grids. It is worth mentioning that the small size of the objects in the aerial imagery dictates having an overlapping window between pieces to ensure no object is being cut into halves and missed in the cut line. Grid cropping can be done with a predefined overlapping amount [80]. In this research, the amount of overlapping window is being calculated with respect to the size of the input images. The height and width of the patches can be interpreted as an arithmetic progression with d_H and d_W as the **common differences of successive members** in height and width sequences, respectively, calculated with the formula 5.1 where the **Number of Cuts in Height** and **Number of Cuts in Width** are $\lceil \frac{\text{Original Height}}{\text{Output Height}} \rceil$ and $\lceil \frac{\text{Original Width}}{\text{Output Width}} \rceil$. Figure 5.1 illustrates this cropping method.

$$
\begin{aligned}
d_H &= \frac{(\text{Number of Cuts in Height} * \text{Output Height}) - (\text{Original Height})}{\text{Number of Cuts in Height} - 1} \\
d_W &= \frac{(\text{Number of Cuts in Width} * \text{Output Width}) - (\text{Original Width})}{\text{Number of Cuts in Width} - 1}
\end{aligned}
\tag{5.1}
$$

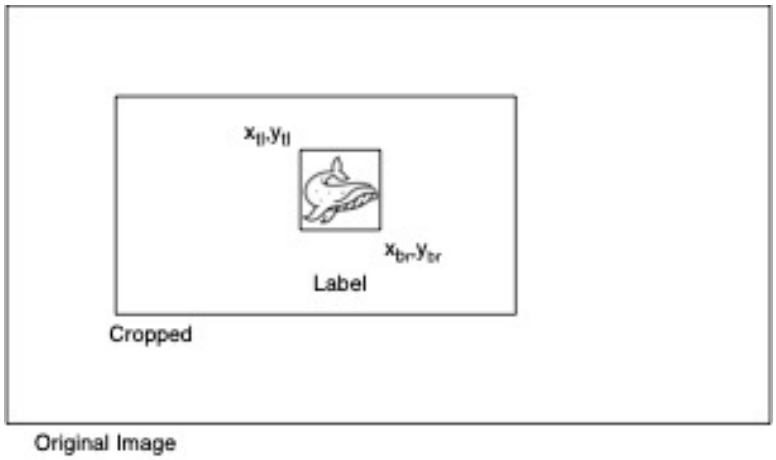


Figure 5.2: The random cropping method

5.2 Random Cropping

Objects comprise only a small fraction of the data, and the majority of data is water. It is desirable to have a model that can handle various contexts around the objects. Hence, another random cropping method also was applied to training and testing to make this possible.

In the random cropping method, the location of the objects is known, and the cropping algorithm crops images with respect to locations and sizes of the objects and having a random amount of contextual pixels around the objects. Figure 5.2 shows the random cropping method. This method is also applied to the training and testing data to have a diverse training set and a fair performance measure.

5.3 Data Augmentation

Training machine learning models using a vast amount of data yields a better generalization than training those on small-scale datasets. However, collecting a large-scale dataset such as ImageNet is not feasible for every problem. The amount of data can be limited for various reasons, including limited real-world examples, a limited number of experts, and a limited financial budget. In some cases, the number of occurrences of a particular event in the real world is limited, such as a rare disease. In other scenarios, the number of human experts or the expense of acquiring a data point can be limiting factors.

Data augmentation or dataset augmentation is a technique for generating fake data from the actual data and including them in the dataset fed to the machine learning model. Data augmentation is a part of the online pre-processing techniques used in this research. Generating fake data does not have a straightforward method for various machine learning tasks since, for some of them, the problem should be solved to some extent to have fake data similar to the actual data. Some methods, such as SMOTE [81], create more training examples synthetically, but the complexity of this data made it seem unlikely for SMOTE to be effective in our context. The interpolated images would not be expected to be realistic, and a more effective approach in future work might be incorporating a strong mechanism for out-of-distribution detection. Because of its effectiveness, data augmentaion became a de facto part of computer vision deep learning pipelines.

Although data augmentation cannot solve the insufficient data problem, it can use a higher potential of the available data. Augmentations are significantly effective and advantageous on small-scale datasets [82, 83]. Transforms and distortions can be used in computer vision tasks as long as they do not destroy the data or change the data class. Different distortions can be used in computer vision tasks that are computationally cheap and do not need the main task to be solved. These transforms include but are not limited to flipping, rotation, affine transformations (spatial transforms), and color jittering, histogram equalization (pixel-wise transforms).

Cropping images creates two kinds of sub-images: **positive** samples and **negative** samples. Positive samples are the sub-images that have objects of interest in them. On the other hand, negative samples are the sub-images that do not contain any objects of interest and should be considered background (water). Negative samples are necessary for the task since only a tiny amount of pixels in images are objects. The data augmentation pipeline used in this project has separate augmentation policies for each group of negative and positive image samples. Suppose an input image contains an object of interest in it. In that case, the image goes through the **positive** image augmentation, and the path for the negative samples is through the **negative** sample image augmentation.

The positive image augmentations include but are not limited to vertical and horizontal, jpeg compression, and histogram equalizations. The transforms for the

negative examples have a longer list, including what is being used on the positive side since some distortions are useful for capturing various types of noise and sea colors. The complete list of data augmentations used in training networks is written in Appendix B. The torchvision and **Albumentations** library [84] were used to build the augmentation pipelines.

The other purpose of data augmentation is to use it as a regularization technique that can be effective in decreasing generalization error. The model capacity is often high to capture the complexities in the data and learn essential features from it, which makes the models prone to overfit [85]. Data augmentations in convolutional neural networks can be more potent than popular regularization techniques such as dropout [86, 87]. While both data augmentation and dropout are effective methods to reduce the chance of overfitting, in convolutional neural networks, dropout is less effective due to two prominent reasons. First, convolutional neural networks are empowered by parameter sharing concepts to reduce the model complexity. So, in essence, these networks need less regularization compared to fully connected networks [85]. Second, in images, the context around each pixel is information-rich so that dropping one pixel cannot be effective as the surrounding context and the spatial correlation is rich enough to cancel the effect of dropout [85].

The third possible benefit of data augmentation is that it can increase the networks' robustness to the input noise and deformations. Neural networks are vulnerable to input noise [88], and adding random noise as a data augmentation technique can act the same as a regularization method to increase the robustness of the model.

5.4 Normalization

Normalization, also known as **feature normalization** or **z-scoring**, is one of the crucial ingredients of training neural networks [89] and refers to removing moments - the mean and standard deviation- of the data from samples. As neural networks started to go deeper, the normalization operation gained more importance. In addition to being a pre-processing step, some variants of it became internal operations of neural networks such as **batch normalization** [90] and **group normalization** [91]. In this research, as a part of the online pre-processing pipeline of training neural networks, the normalization step normalizes the input images after data augmentations.

Chapter 6

Object Detection

In this chapter, we will define object detection as a computer vision task and describe the deep architecture we used. We discuss training and evaluation of this network later in Chapter 8.

6.1 Object Detection Computer Vision Task

The **object detection** task can be considered the successor of two other computer vision tasks of **image classification** and **image classification + localization**. Image classification is the task of predicting a class value for an input image. The class value belongs to the object that typically covers the majority of the image, and there is no other object to be considered in the input image. Image classification + localization not only predicts the class value for an image but also predicts the exact location of the primary object in the input image. Localization predicts four values representing four corners of a bounding box circumscribing the object. Both of the aforementioned tasks have a only one object to recognize Object detection is a generalization of the latter task without a pre-defined number of objects in the input images. Object detection has several challenges in addition to the ones the other two have to tackle. Two of the most important ones are the presence of objects in multiple scales and designing an architecture that can handle variable length outputs corresponding to the number of objects present in the input image. Figure 6.1 depicts these tasks.

When trying to find animals and objects in aerial imagery, object detection was the first approach that we tried. Object detection models are designed to find objects of interest in images and suggest bounding boxes around them. For this approach, we utilized the **Faster Region-based Convolutional Neural Network** or **Faster R-CNN** architecture.

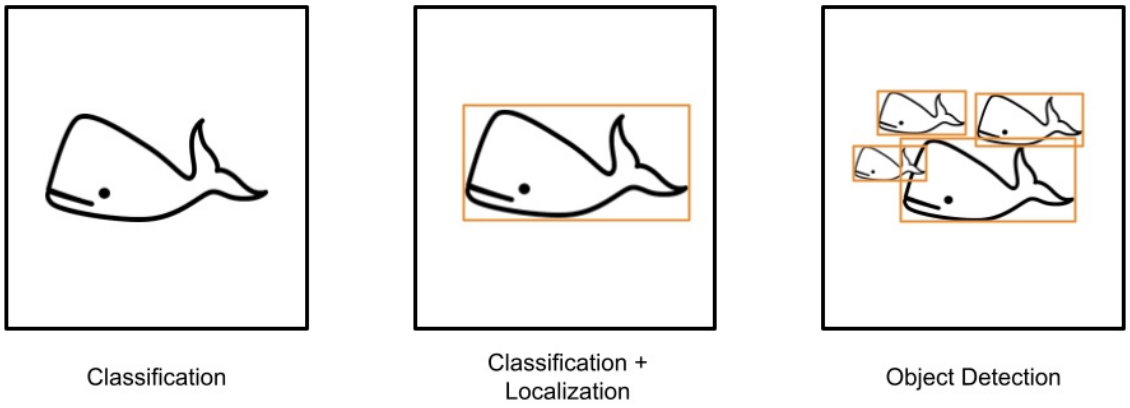


Figure 6.1: Object detection and its forerunner computer vision tasks

6.2 Faster R-CNN

Faster R-CNN is the third member of the R-CNN object detection architecture family after Fast R-CNN [24] and R-CNN [22]. Faster R-CNN enabled end-to-end network training and evolved into a more complex and more accurate architecture through several upgrades since its introduction. In this section, we briefly discuss the version that we used for the project.

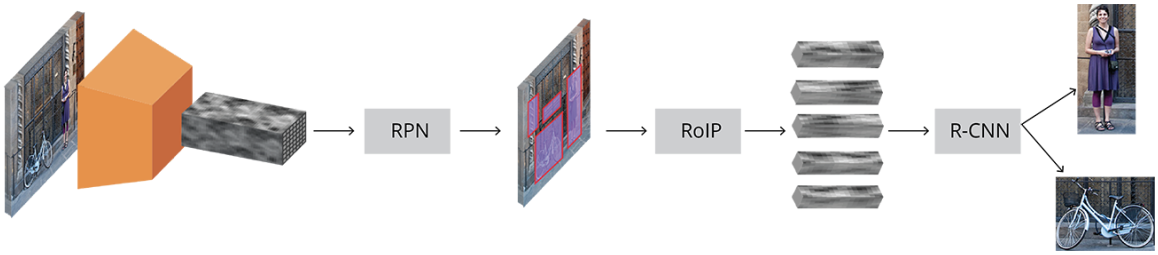


Figure 6.2: Faster R-CNN Object detection pipeline. The image is taken from [3]

The Faster R-CNN pipeline has several components that we are going to explain in detail later in this chapter. The first component is the base network or the **backbone**, a convolutional neural network for feature extraction. The key idea is to use a pre-trained classification network to extract a rich convolutional feature map. This component utilizes the Feature Pyramid Network (FPN) approach to deal with the multiple scale problem of the object detection vision task. The second component is the **region proposal network** that takes the extracted convolutional map and does the first step of object detection with two labels of **background** and **object** and

passes the objects it found to the next component as the proposed regions. Region of Interest Pooling (RoIP) is the next component that takes the proposed regions, performs a max-pooling operation, and prepares them for the R-CNN head that classifies these fixed-sized convolutional maps. Finally, classifications and bounding boxes are being processed to be reflected on the original input image as a post-processing step. Figure 6.2 shows the pipeline of Faster R-CNN, and the components are discussed in detail below.

6.2.1 Backbone

Backbone is the base network for feature extraction from the input image. As mentioned before, the primary notion is to use a classification network to extract beneficial useful features. Initially in the original implementation, ZF [92] and VGG [39] were used as the backbone network of Faster R-CNN, but we used the ResNet-50 network.

The convolution operation keeps the relative locations of the activations fixed with respect to the input features. In other words, a certain activation that is related to the presence of a certain feature in the input image appears in the location of that feature. For example, an edge detection filter has an activation pattern that is prominent in the location of edges in the input signal. This locality characteristic in the convolutional operation is called **translation equivariance**, i.e. the same pattern translated to another location in the image will elicit the same response translated by the corresponding amount. Translation equivariance is the key concept in using a convolutional neural network to extract useful features for object detection. For this project, we used ResNet-50 network as the backbone network. ResNet-50 consists of five residual blocks that each block shrinks the spatial resolution of the input to half that the whole architecture makes the input signal 32 times smaller in each x and y axes. The features can be extracted from the last convolutional layer of ResNet with semantically strong features or utilizing a Feature Pyramid network, which we used.

6.2.2 Feature Pyramid Network

Feature pyramids are advantageous components in object detection, but they were usually not included in deep object detection networks due to their memory and

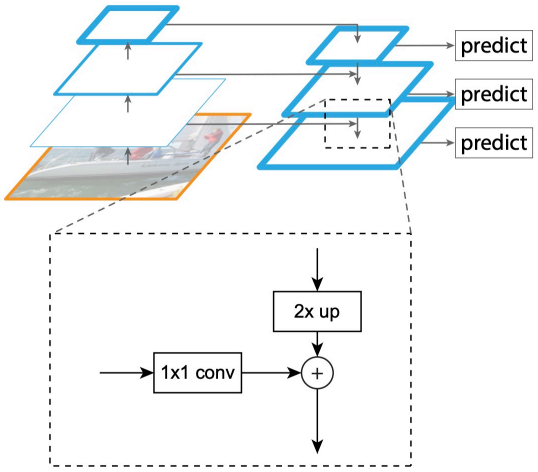


Figure 6.3: Building block illustration of the feature pyramid network. The image is taken from [4]

compute-intensive nature until the introduction of the **feature pyramid network** (**FPN**) [4]. One of the challenges that object detection models face is the presence of objects with multiple scales, and the feature pyramid network is an effective way to introduce input in different scales to the network with a set of low-cost operations. The feature pyramid network is an add-on network for the feature extractor backbone of Faster R-CNN, which is a ResNet. The goal is to merge high-resolution but semantically weak features with low-resolution and semantically powerful features.

The deeper the layers are in the feature extraction network, the lower spatial resolution and semantically powerful features they have, and FPN takes advantage of this hierarchy of features. FPN can be divided into three pieces: the **bottom-up pathway**, the **top-down pathway**, and **lateral connections**. The bottom-up pathway goes from the input image to the last convolutional layer in the backbone network, and the top-down pathway comes back from the last convolutional layer backward to the input. The first layer in the bottom-up pathway is the last layer in the top-down pathway and vice versa. Figure 6.3 illustrates the FPN architecture.

The output of FPN is a set of hierarchical feature maps to be fed into the next component, Region Proposal Network, for a scale-aware region proposal.Each ResNet block is considered as a level. At each level, the feature map is being upsampled using the nearest neighbor upsampling algorithm by a factor of two to compensate for the downsampling effect of the ResNet blocks. In this case, the spatial resolution

matches the resolution of the feature map of the previous layer (in bottom-up view). Then using a 1×1 convolution operation, the two upsampled finer and coarser feature maps are being added to produce a feature map for the corresponding level. This sequence of upsampling, the 1x1 convolution operation, and the addition is called the lateral connection in the context of the feature pyramid network. As there is no extra block beyond the fifth one, the lateral connection for the last level only has a 1×1 convolution operation.

6.2.3 Anchors

The object detection task predicts a variable-length sequence of bounding boxes with their corresponding classification scores. The prediction has a variable length because the number of objects presented in images is not predefined. Using anchors is a method to convert the variable-length predictions to fixed-length ones. Anchors (also known as anchor boxes) are reference boxes that are being generated at every point in the convolutional feature map. Anchor boxes have different but predefined aspect ratios and scales to cover objects that can be presented in multiple scales. Anchor boxes are the reference regions for the next module (RPN) to find the best candidates for the final classification. We utilized three aspect ratios of 0.5, 1.0, 2.0, corresponding to a vertical rectangle, a square, and a horizontal rectangle, respectively, with five scales of 32, 64, 128, 256, and 512, totaling 15 different combinations at each point in the feature space. Considering an input image with the size of 1200×800 pixels and the fact that the backbone of the network decreases the spatial resolution by a factor of 32 in each direction, it has approximately 14250 anchors ($38 \times 25 \times 15$).

6.2.4 Region Proposal Network

In two-stage object detection algorithms, the underlying idea is to have a sliding window over every possible position on the image and classify the patch using a classification network. This approach is not practical due to the high volume of computations needed. Therefore, several methods have been developed to shrink the search space for the classification. **Region proposal network (RPN)** is a building block of the Faster R-CNN architecture that takes the convolutional feature map extracted by the backbone and generates (proposes) regions that contain objects.

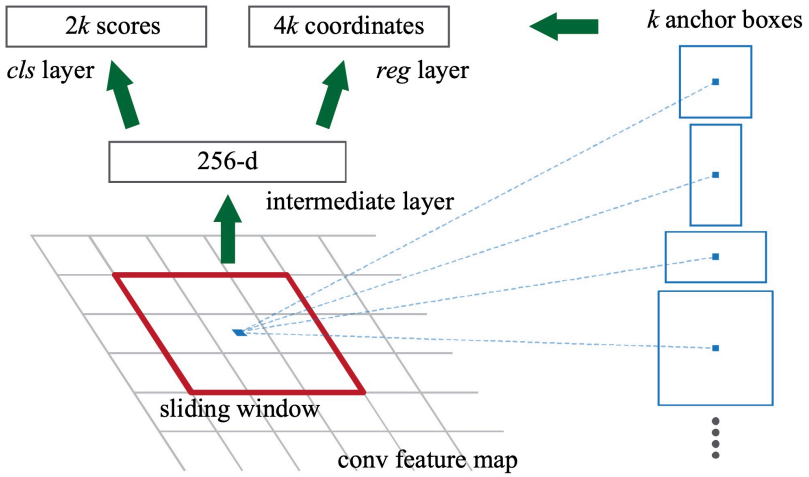


Figure 6.4: Region Proposal Network. The figure is taken from [5]

RPN is a weak object detector, and it is the first stage of the detection steps of Faster R-CNN before the primary classifier of the network. RPN has two heads of classification and regression. The classification head of RPN is a binary classifier of object versus background. The regression head optimizes the offset of each patch from the anchor boxes with four values corresponding to four values needed to encode bounding boxes. Two loss functions should be optimized in this network: binary cross-entropy loss for the classification head and smoothed L1 loss for the regression head. Figure 6.4 illustrates this network.

After getting proposed regions, they are filtered by three criteria: objectness score, region size, and non-maximum suppression. The objectness score is the RPN classification score, marking the region as an object instead of the background, and it should be above a certain threshold. Also, a threshold on the minimum of the proposed region size helps eliminate tiny regions that are practically false positives. In addition to these, the non-maximum suppression algorithm (described next in Section 6.2.5) keeps only one candidate from a set of overlapping proposed regions.

6.2.5 Non-Maximum Suppression

Non-maximum Suppression (NMS) is a simple algorithm to reduce the number of overlapping bounding boxes and regions. In the object detection task, it is possible to have several overlapping detections with different classification scores. NMS tries to eliminate the overlapping bounding boxes and keep the best candidate among those.

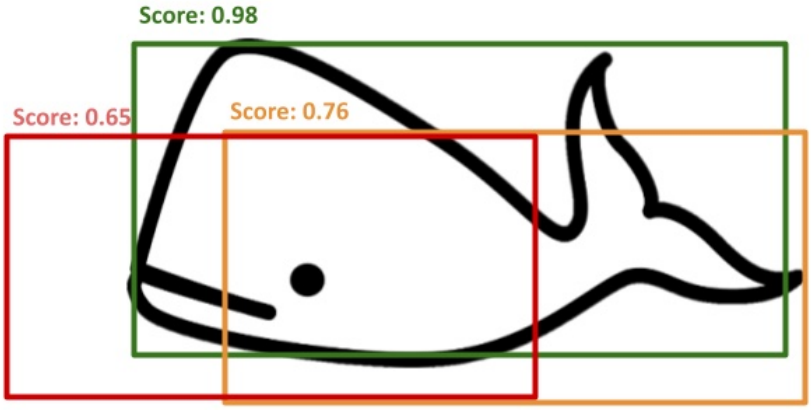


Figure 6.5: Three predictions for one object with different confidence scores

The algorithm first sorts the candidate bounding boxes by the confidence score, then eliminates the bounding boxes that have an IoU (defined in Appendix E.1) of over 0.7 with another one with higher confidence. For example, in figure 6.5, the green box with higher confidence has a high overlap with orange and red boxes; hence, only the green one will be kept using NMS. In the presence of multiple classes, the process will be the same for each class, individually.

Using the classification confidence score as a reference for elimination can be complicated. Deep neural networks often produce incorrect but overconfident results when the input is out of distribution [93]. Therefore, to have a working NMS algorithm, this phenomenon should be taken into consideration.

6.2.6 Region of Interest Pooling

Region of Interest Pooling (RoIP) is another module of Faster R-CNN and its predecessor, Fast R-CNN, that takes the regions that RPN proposes and outputs feature maps with fixed spatial size. The regions from RPN do not have fixed sizes, and RoIP mitigates this by applying a max-pooling operation in a window method fashion. A max-pooling of size HxW (for example, 7×7 is the size used in this project) picks the maximum values from the regions with a spatial resolution of $h \times w$ (the height and width of the proposed regions, respectively)by first slicing that into squares with the spatial resolution of $\frac{h}{H} \times \frac{w}{W}$ squares and then picking the maximum value in those squares. For example, assume there is a region with a spatial resolution

of 56×63 with n channels, a RoIP module with a pre-defined output resolution of 7×7, slices that region and applies max-pooling on volumes of resolution 8x9 (height and width)

6.2.7 Region-based Convolutional Neural Network

Region-based Convolutional Neural Network (R-CNN) is the final stage of the Faster R-CNN detection pipeline, where the model predicts objects based on the convolutional feature maps derived using RoIP. R-CNN has two fully connected networks for classification and regression tasks. The classification task is for predicting the class of the region, and the regression task is for finding the four corners of the boxes. The loss functions for the classification and the regression are cross-entropy loss and smoothed L1 loss, respectively. These two losses are being combined by summing up these two loss terms without any weights.

Chapter 7

Semantic Segmentation

In this chapter, first, we will define semantic segmentation as a computer vision task, and second, we will describe Deeplabv3, the deep architecture we used for this approach. We discuss training and evaluation of this network later in Chapter 8.

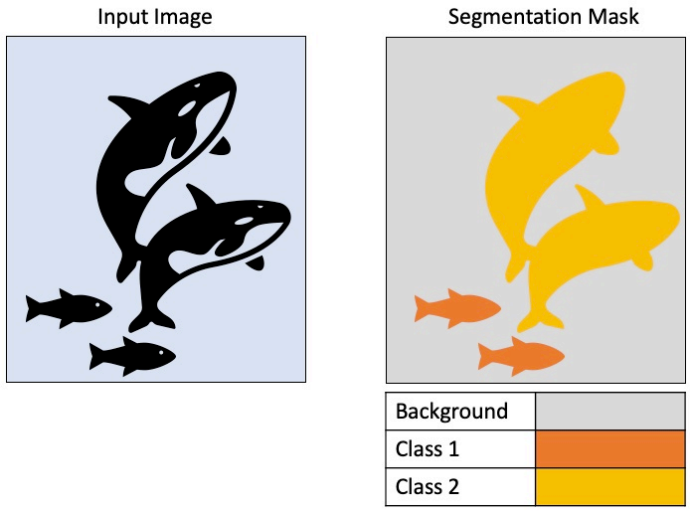


Figure 7.1: Semantic Segmentation Task

7.1 Semantic Segmentation Computer Vision Task

Semantic segmentation is another computer vision task in addition to the three tasks discussed in section 6.1. Semantic segmentation is one step closer to comprehensive scene perception — the ultimate goal of computer vision. In this task, the processing algorithm outputs a dense pixel-wise prediction for the input image. Each pixel in the input image gets classified. That is, the predictions are two-dimensional (one channel) signals with the exact spatial resolution as the inputs containing information of pixel-level classification. For example, in figure 7.1, the input image contains four objects, two of them are from one class and the other two from another class of objects. The

semantic segmentation mask of this image is a two-dimensional signal with a class id for each pixel. There is no distinction between instances of a class and all instances of a class, whether overlapping or not, get the same id in the predictions.

As our second approach besides object detection , we tried DeepLabv3 as a state-of-the-art semantic segmentation fully-convolutional architecture to find animals and other objects in the aerial imagery.

7.2 DeepLabv3

DeepLabv3 [46] is an upgraded version of earlier DeepLabv2 [8] and DeepLab [45] semantic segmentation networks. Despite its relatively simple architecture and using well-known building blocks, it is one of the best performing semantic segmentation architectures. Similar to previous models in the DeepLab family, DeepLabv3 tries to overcome two primary challenges. The first challenge is the undesirable reduction in the feature map's spatial resolution after consecutive convolutional and pooling operations. These operations reduce the resolution as the input goes deeper in convolutional networks. This phenomenon makes the networks capable of learning more abstract features in deeper layers, which benefits a variety of machine learning tasks that dimensionality reduction is an asset. The second challenge is that objects are present at different scales in images. This can be challenging due to the fact that objects at different scales have different prominent features. In this section, after giving an overview of building blocks, we will briefly discuss the architecture and approaches used in DeepLabv3 to tackle the challenges mentioned earlier.

7.2.1 Dilated Convolution

In convolutional neural networks, **receptive field** is the size of a grid from the input signal that is responsible for computing a feature in the feature map. In other words, receptive field is the area that a convolutional filter considers for computing a feature. A higher receptive field means considering more context to compute a feature and making predictions. Conventionally, stacking more convolutional layers increases the receptive field, which is vital for feature extraction. With the advent of more complex and advanced deep learning models, the receptive field increased to the extent that most recent networks have a receptive field that covers the whole input image [94].

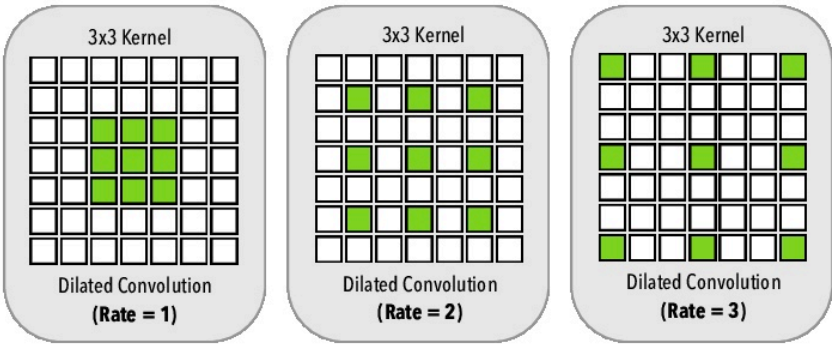


Figure 7.2: Dilated convolution kernel with various dilation rates. The figure is taken from [6]

Repeated convolutional layers and pooling operations increase the receptive field and decrease the feature map's spatial resolution. **Deconvolution** is a method to recover the spatial resolution; however, DeepLab models use **dilated convolution** to preserve the resolution.

Dilated convolution (also known as **atrous convolution**) is a variation of convolution operation in which the convolutional filter is upsampled. Upsampling a convolutional filter means enlarging it. In dilated convolution, the filter is upsampled with inserting zeros between consecutive values in the filter. This approach increases kernel size without adding extra learnable parameters. Dilated convolution in 1-dimensional discrete signals is defined as Equation 7.1 (Note that the non-mirrored notation of the convolution operation is used). According to the definition, the kernel gets multiplied by the signal in every r step. The r is called sampling rate or dilation rate. Two intermediate values in the kernel correspond to two values in the input with the offset of $r-1$, or in another interpretation, $r-1$ zeros are inserted between two consecutive values of the kernel. In a standard convolution filter, the $r = 1$. Figure 7.2 depicts a 3×3 convolutional kernel with several dilation rates. Dilated convolution makes it possible to explicitly control the receptive field with the dilation rate without stacking several layers and additional parameters. Figure 7.3 illustrates a 3×3 convolutional filter with different dilation rates and their effect on the receptive field.

$$y[i] = \sum_{k=1}^{K} x[i + r \cdot k]w[k] \tag{7.1}$$

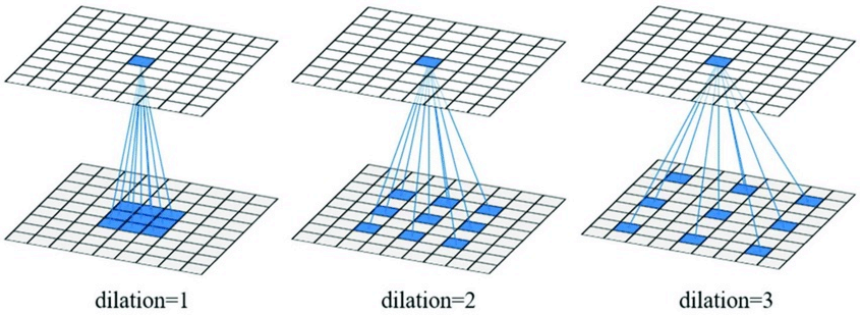


Figure 7.3: Receptive field, considering various dilation rates. The figure is taken from [7].

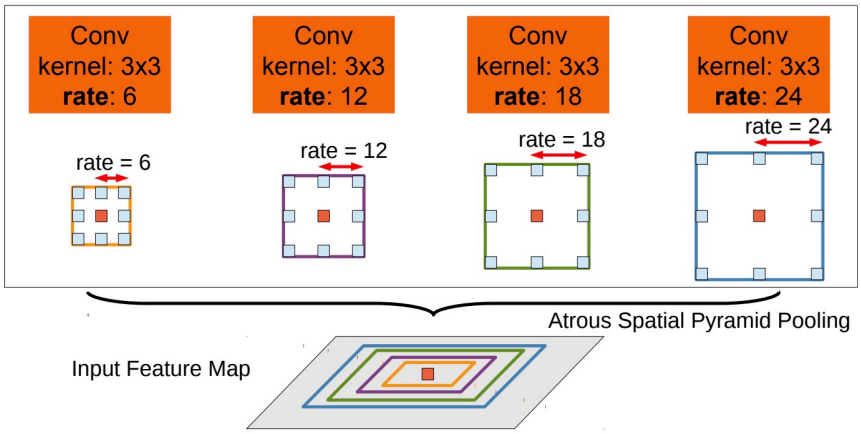


Figure 7.4: Atrous Spatial Pyramid Pooling. Multiple filters with different field of views depicted in different colors are being applied to classify the center orange pixel. Figure is taken from [8].

7.2.2 Atrous Spatial Pyramid Pooling

The second challenge that DeepLabv3 attempts to overcome isthe presence of objects at multiple scales.

Deep neural networks are capable of generalization in this manner; however, explicit consideration of this challenge can benefit the generalization. **Spatial Pyramid Pooling (SPP)** [23] is a technique that was developed to solve this challenge in object detection. SPP technique increased the model's robustness to the input image deformations while the number of computations was reduced. Inspired by the success of SPP, an Atrous variant of that called **Atrous Spatial Pyramid Pooling**

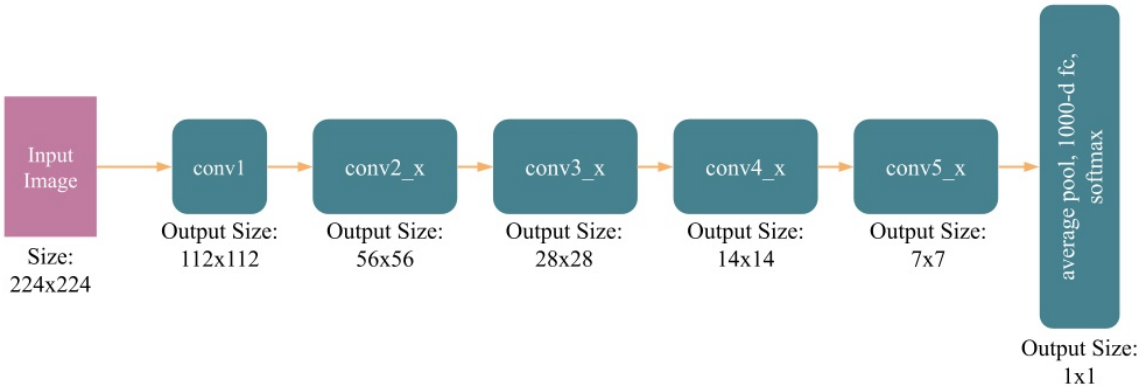


Figure 7.5: ResNet Blocks and their output sizes

(**ASPP**) was used in DeepLabv3. In ASPP, multiple parallel atrous convolutions with different dilation rates are being applied on the feature map extracted from the input image. These parallel operations are getting fused together by concatenation and a 1×1 convolutional filter to form the final logits. Figure 7.4 illustrates the ASPP module field of view in different sampling rates.

7.2.3 Architecture

After the introduction of two main components of DeepLabv3, it is simple to demystify the end-to-end architecture. Repurposing a pre-trained ResNet architecture is the backbone of DeepLabv3. Considering the modular design of ResNets shown in figure 7.5, each block shrinks the spatial resolution to $\frac{1}{2}$ of the resolution of its input , resulting in downsampling of ratio 32 at the end of the last convolutional layer. In DeepLabv3, the stride of the last two convolution layers is set to 1 to prevent the resolution decimation, and dilated variants replaced the standard convolutions in those layers. This modification keeps the spatial resolution to $\frac{1}{8}$ of the input signal at the last convolutional layer. This convolution feature map is the input to the ASPP module to calculate the logits of the pixel-level classification. Then these logits are being bi-linearly interpolated to the size of the input image that is eight times bigger than the logits to get the class predictions with an equal size to the input image. These logits are then being fed to a softmax function to get the belief map to produce the final pixel-level class label.

Chapter 8

Training Networks and Evaluation

In this chapter, we discuss the approaches we have taken for data preparation, training networks, and evaluating the models. Appendix C has the full list of hyper-parameters we used for training our models

8.1 Overview

We prepared data for training models. Then we trained our models on the prepared datasets using Train and Validation sets. We evaluated the models' performance on comparison sets, both numerically and using human experts afterward. The purpose of this evaluation is to find the best-performing model and find a meaningful relationship between numerical and AI-in-the-loop evaluation.

8.2 The AI-in-the-loop evaluation

To evaluate the effectiveness of a prediction tool for helping a human create a larger dataset, we first need to understand how human experts currently go about labeling datasets. Based on extensive discussions with—and observations of—expert annotators, we found that the real-world scenario of image annotation by a human expert can be divided into three phases. In phase 1, the expert labeler chooses a batch of images, goes through them manually to find objects of interest. The labeler is unaware of the images' content, so some of the chosen images possibly contain objects of interest. However, it is not guaranteed to find animals in the batch. In the case of a successful search, the labeler puts the useful images aside to annotate them. In phase 2, the labeler prepares the data for labeling in the labeling software. Finally, during phase 3, the expert labels the images and exports the annotations.

This is the process we wish to accelerate or improve, and thus we used these observations to to develop the evaluation procedure, as we will explain shortly. We

will also provide further details on these phases in each condition. We timed these phases in each condition to observe the effect of the machine learning model's aid on object annotation.

We conducted an AI-in-the-loop evaluation that included each of the two approaches that we used to try to detect animals in the aerial imagery, i.e. object detection (Chapter 6) and semantic segmentation (Chapter 7). The experiment's goal is to find the effect of each of the two approaches (object detection and semantic segmentation) on the amount of time needed to find the objects of interest and annotate the images in order to help to create a bigger dataset. That is, a successful system *will effectively allow us to bootstrap the creation of a large dataset.* This evaluation also makes it possible to select the superior model or combination of models for accelerating labeling.

8.2.1 Experiment Protocol

We conducted the AI-in-the-loop evaluation in the form of an experiment. As we mentioned earlier, the goal of this experiment is to observe the effect of a computer vision model as a helper tool on annotating and creating bigger datasets starting from a limited amount of labeled data.

The experiment has two types of evaluations: **metrics** and **time measurements**. Metrics allow us to evaluate models' performance on finding objects of interest, their accuracy, and how they help detect animals. Time measurements assess how helpful the models are for annotating more images by the expert human labelers.

The experiment has three conditions: The control condition (fully manual), object detection model as a helper, semantic segmentation model as a helper. In each condition, the expert labeler gets a set of images; we call this set of images comparison set, that some have no animals in them, and some contain animals. Then the labeler annotates images in the comparison set in three phases that we discussed earlier. More details will be given on each condition in the Section 8.4 on the experiment, training the models and evaluating them.

Each condition's comparison set contains 61 images from the 1544 images we have in the dataset. Between the experimental conditions, the comparison sets should be different. If we keep the comparison set the same among all the conditions, the

human labeler's performance can vary. If the labeler goes through the same images several times, the latest rounds of checking become faster, and the time measurements become unfair. Furthermore, the comparison sets should not be drastically different between conditions. Overall appearance and the environmental factors in the images can affect the performance of the labeler in detecting objects; therefore, it causes unfair comparisons between conditions. We prepared comparison sets without any duplicated images between the conditions, but they have similar appearances. We discuss this process in Section 8.3.

Since we had no data samples beyond 1544 images, neither labeled nor unlabeled samples, we selected the comparison set images from the primary dataset. The labelers have seen those images and already labeled them, we applied three modifications to them to decrease the chance of recalling some samples. We used random horizontal flip and vertical flip on the images in the comparison sets. Not all images have objects of interest in practice. To simulate a real-world scenario, we need to erase some of the images' objects. We tried two inpainting methods: Fast Marching Method based inpainting [9] and Navier-Stokes based inpainting [95]. **OpenCV** library covers both of these methods. These methods need a mask for the areas to inpaint, and since the ground truth masks we have are noisy, the resulting inpainted image sections had residues. This could affect the experiment, so we decided to remove the objects manually, using photo editing software. Figure 8.1 illustrates an example of removing a `Humpback` whale from the image using inpainting and photo editing software. Removing the animal manually, leaves no residue and makes the images look naturally empty.

We take two measures to have comparable results between conditions. Object detection model predicts bounding boxes while semantic segmentation's predictions are pixel-level masks. For time measurements, we convert the segmentation predictions from pixel-level to bounding boxes (to read about this procedure, please refer to Appendix D). Secondly, for time measurement on comparison sets, we omit two labels that were not the primary intention of the projects. Two labels of `Object` and `Artifact` do not have a comprehensive labeling scheme over the dataset, and labeling them makes the time measurements inaccurate and subjective. Initially, the expert

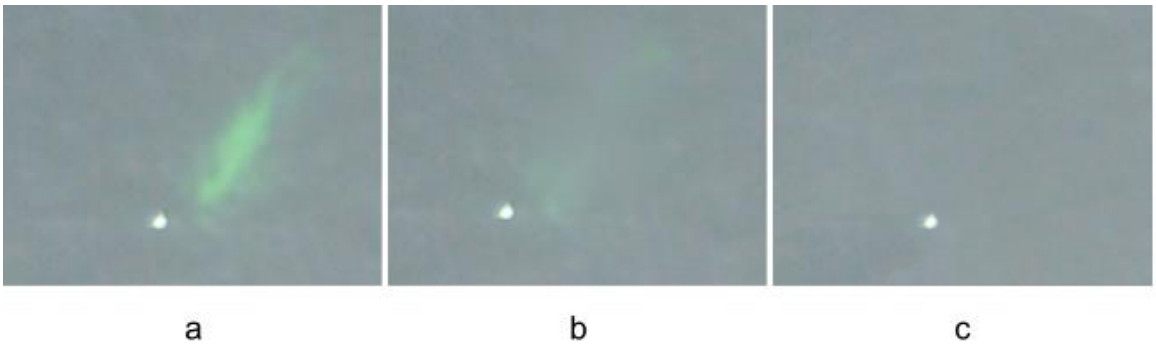


Figure 8.1: An example of using inpainting techniques for removing objects from images. (a) is the input image, (b) is the inpainting result using [9] and (c) is the result of manually removing objects with photo editing software.

labeled them when they were similar to some objects in the main object classes. However, we kept them for the training and evaluation of the models using metrics since more data samples can make better detection models.

Another point is that the models have different crop sizes of input images. This means they need different input image resolutions for training and inference. Since all the predictions are automatically adjusted to the original image sizes, the predictions and images for the conditions are comparable.

8.3 Data Preparation

In this section, we will first develop a visualization to let us observe the dataset all at once in a grid, making it easier to observe and discuss its characteristics. Based on these findings, we will describe our approach to prepare the data for each experimental condition.

8.3.1 Clustered Dataset Grid

Observing the image dataset all at once can reveal information about the dataset that can be utilized in the research. We used image grids, popular in the field of computer vision, to do this.

We first built a dataset grid sorted by filenames, and the appearance suggested the presence of underlying clusters. One possible approach to group the images by visual similarity is to use unsupervised techniques to cluster images based on the extracted

features. We used a pre-trained ResNet101 image classification network to extract features.

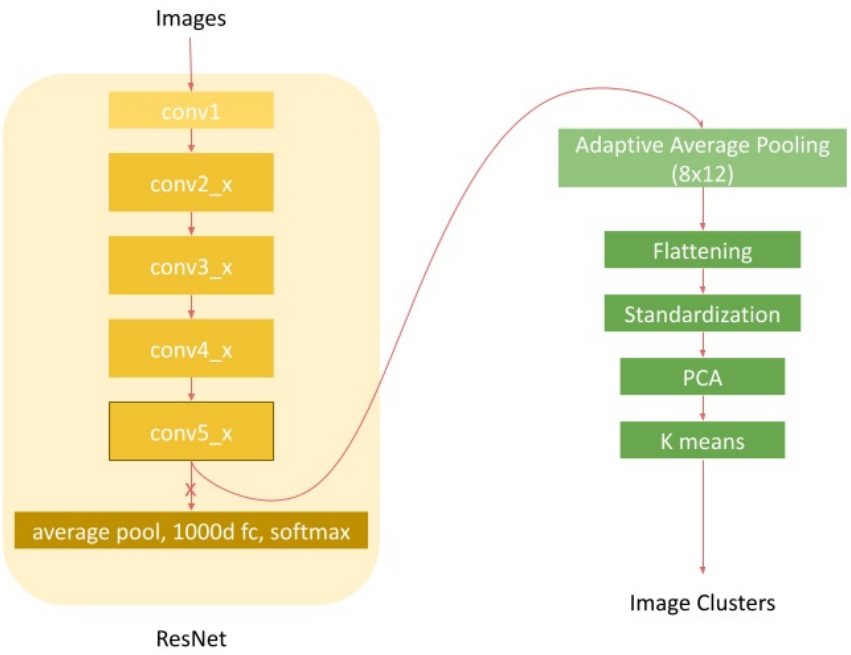


Figure 8.2: The image clustering pipeline

Extracting Visual Feature Representations

Since the introduction of Residual Neural Networks (ResNets) [96] in 2015, they are widely used in several tasks of computer vision. ResNets are the backbone of object detection, and semantic segmentation models such as Faster R-CNN [5] and DeepLabv3 [46] since they can extract rich feature representations from images.

To extract convolutional feature representations of the images, we downsampled them from 4912×7360 pixels to 800×1200 pixels. Then we fed the downsampled images into a ResNet101 to get the convolutional features tensor of size $(2048 \times 25 \times 38)$ from the last convolutional layer of the network (`conv5_x`). Then we applied an adaptive average pooling of size 8×12 (proportional to the 800×1200) to them, making a tensor of the size of $2048 \times 8 \times 12$, giving 196608 features after flattening.

Image Clustering

We then reduced the 196608 flattened features to 50 principal components using PCA algorithm. Since PCA algorithm is sensitive to high variations in the input data, we did a standardization step before feeding the features to PCA. Then we fed the matrix of 1544 (number of images) $\times$50 (reduced features) to K-means clustering algorithm to discover underlying clusters, setting the number of clusters hyperparameter to 10. Figure 8.2 is the pipeline of the process of clustering. Figure 8.3 (partially) illustrates the clustered dataset image grid. In order to fit on the page, the grid only shows the first five clusters.

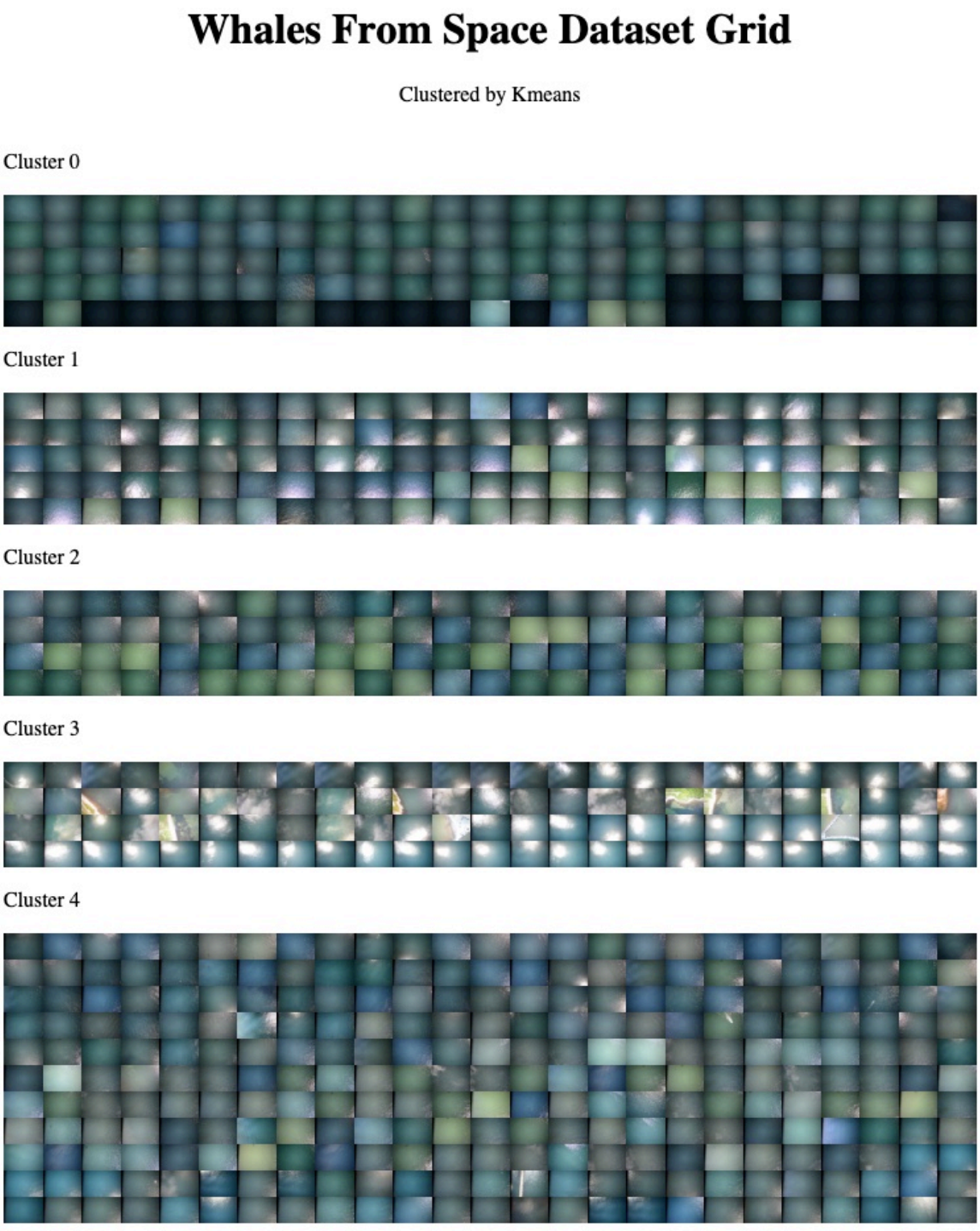


Figure 8.3: The grid of the dataset images

8.3.2 Data Description

For the purpose of training models and evaluating them, we partitioned the data into four subsets: **Training**, **Validation**, **Comparison**, and **Test** sets in each of the conditions of the experiment. Figure 8.4 is a pie chart of the portions of each subset in the dataset that was used to develop and evaluate models. The training set was used to train models, and the validation set was an auxiliary set of data to determine when the model converges and to measure performance. We reserved the test set for the last evaluation before the deployment of the model. Finally, the comparison set was used for evaluating models with the help of human experts.

The most critical subset in the data preparation is the comparison set that should be different for each condition in the experiment to minimize the human expert's memory effect. Comparison sets of different conditions should be similar to each other to measure the labeling performance fairly.

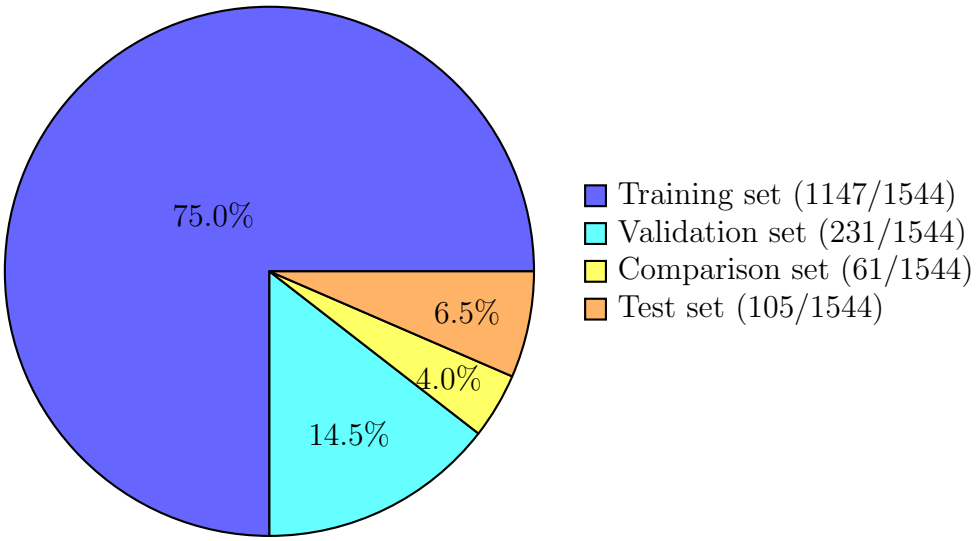


Figure 8.4: The pie chart of the dataset subsets with their percentages and the number of samples in each of them

The number of data samples was a limiting factor both for training and evaluation. The law of large numbers suggests that when sample size grows, the mean and the standard deviation of the samples get closer to the actual mean and standard deviation of the population. In other words, having more samples means a better understanding of the data. One of the notions that can happen in a low number of samples is the

concept of selection bias that makes the data samples not representative of all the samples in the dataset space. Selection bias can be reduced with proper randomization of data. In addition to the problem of limited data, the presence of a human expert in the process of model evaluation is another challenge that we had to consider. Given the diversity manifested in our dataset and the low number of available samples, and the presence of a human expert, we introduced our clustering-based data preparation scheme. We developed an approach similar to the K-fold cross-validation to prepare the dataset for training and evaluating different models

8.3.3 K-Fold Cross Validation

K-Fold cross-validation is a model evaluation technique that is an improvement for the **holdout** method of model evaluation. In the holdout method, the dataset is being partitioned into two subsets of training and testing so that the model is being developed on the training subset and its performance on the test set is the measure of the model's performance of prediction on the dataset. In the k-fold cross-validation method, the data is being partitioned into K equally sized partitions, and in each model development and training, one of the K partitions is being used in the model evaluation while the rest of the (K-1) partitions are being used in the training process of the models. Figure 8.5 is a visual demonstration of k-fold cross-validation. It should be noted that the size of each partition should satisfy the condition of K$\times$(size of the partitions)$\leq$ dataset size so that the evaluation of the model can hold validity since each fold has to have no overlap with any other fold. Similarly, in this experiment, we want to keep the comparison sets similar to each other between experiment conditions; so, the number of conditions can be any number $\leq \frac{\text{dataset size}}{\text{comparison set size}}$.

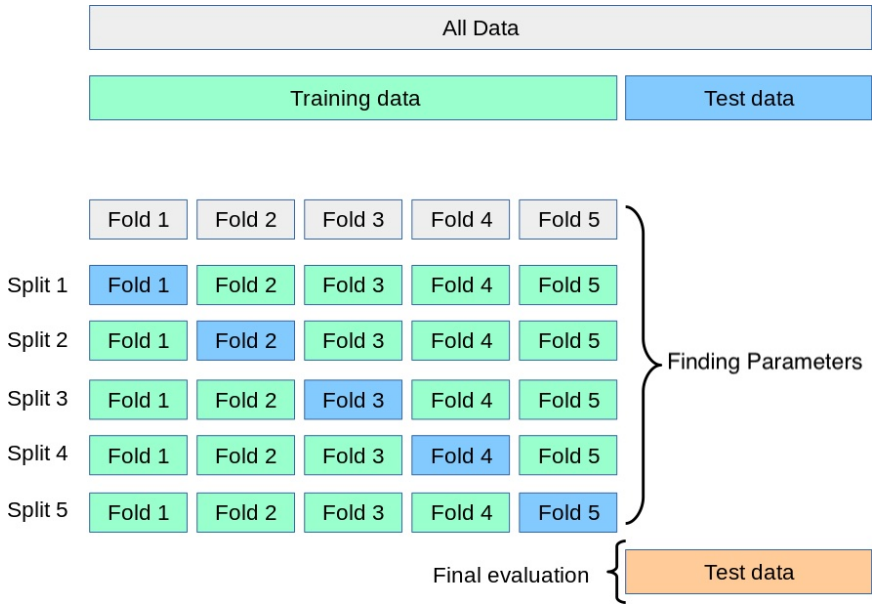


Figure 8.5: Taken from *Cross-validation on scikit-learn's Documentation*[1]

8.3.4 K-Fold Comparison Set Dataset Splitting

To have a data preparation scheme for several experiment conditions while having a comparison set, we devised four rules that should be satisfied as below. We borrowed the first three rules from set partitioning definition.

1. In each condition's dataset, there is no empty subset.
2. In each condition's dataset, the intersection of the subsets are empty sets
3. In each condition's dataset, the union of the subsets is equal to the set of the total dataset
4. The intersection of the comparison sets between folds is an empty set.

Similar to k-fold cross-validation, we devised our approach to split the project's dataset into the four subsets discussed earlier. To uniformly distribute the data samples between the four subsets, we used convolutional feature maps and clustering similar to the dataset clustered grid in Section 8.3.1.

[1] https://scikit-learn.org/stable/modules/cross_validation.html

We are not claiming that there are any true underlying clusters in the feature map that we possibly could discover. On the contrary, clustering in high-dimensional space is a hard task [97] and the purpose of clustering here is to be used as a helper tool to uniformly distribute data among the subsets with respect to their visual feature representations. The number of images in each of ten clusters are plotted in figure 8.6.

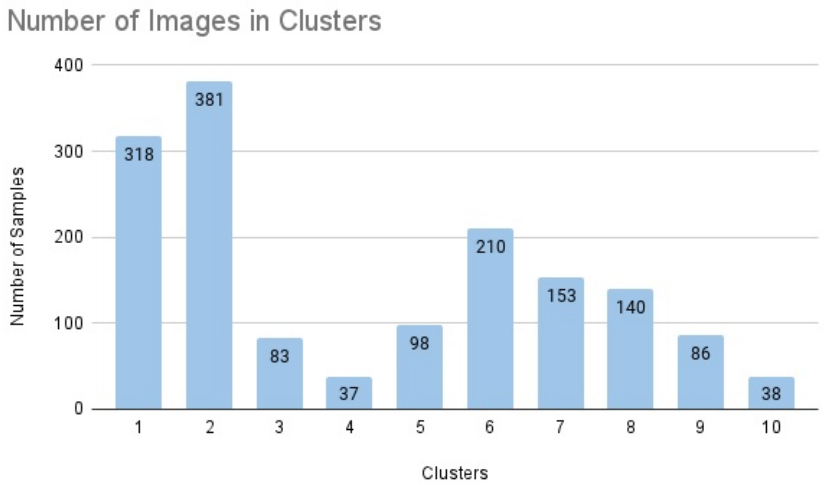


Figure 8.6: Number of images in each of ten clusters

Cluster Verification

The purpose of clustering images was to find groups of similar images based on their latent convolutional features extracted using a pre-trained ResNet. While we have no ground truth for the number of clusters nor the true cluster label for each image, the environmental tags can be useful to empirically verify our method of clustering. We have environmental tags for the majority of the images and as figure 8.7 shows, in each of the ten clusters, there are visible spikes in the categories that make it possible to name each of them as the following list:

- **Cluster 1:** Calm seas with Low glare
- **Cluster 2:** Calm seas with Low glare
- **Cluster 3:** Rough seas with High glare

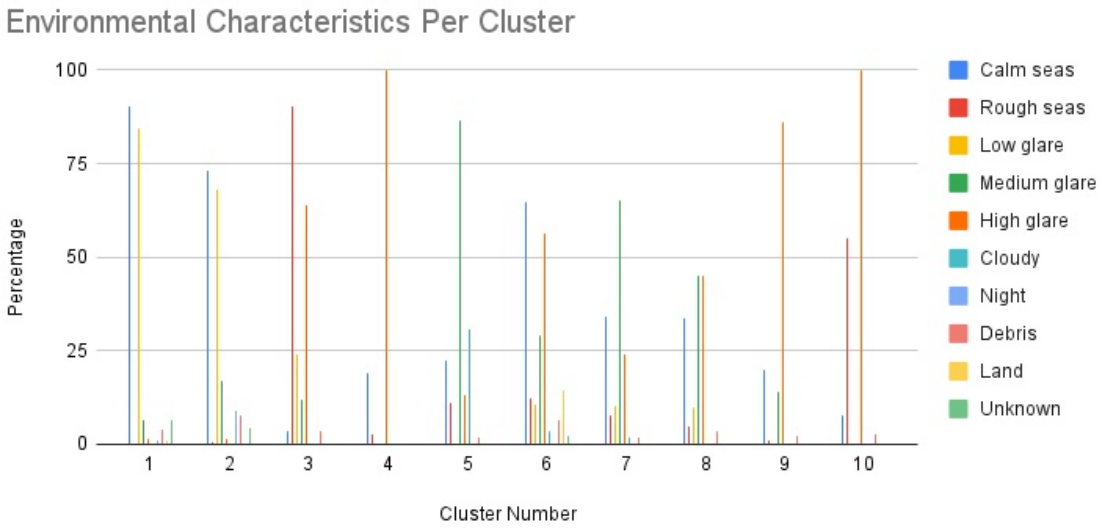


Figure 8.7: The percentage of each environmental tag in each cluster

- **Cluster 4:** Calm seas with High glare
- **Cluster 5:** Medium glare
- **Cluster 6:** Calm seas with Medium to High glare
- **Cluster 7:** Calm seas with Medium glare
- **Cluster 8:** Calm seas with Medium to High glare
- **Cluster 9:** Calm seas with High glare
- **Cluster 10:** Rough seas with High glare

We would like to emphasize that having similar clusters does not harm our use case since we are sampling from each cluster with a pre-defined percentage so that similar clusters will have a greater share similar to having a cluster comprised of those similar ones.

Data Sampling

We have the ratio of each subset of the dataset from Data Description (Section 8.3.2): 75%, 14.5%, 4%, and 6.5% for Training, Validation, Comparison, and Test sets, respectively. It is possible to form the k-fold comparison set using clusters and ratios

from previous steps. We wanted to prepare 10 folds that are equivalent to 10 experiment conditions, but we used three folds at the end. To have distinct, but similar comparison sets, we first sampled $K = 10$ times the comparison ratio (4%) from each cluster without replacement and assigned them to their proper folds. This forms different comparison sets for each fold (experimental condition), visually similar because of using clustering. The other 96% of the data (after sampling comparison sets) in each fold was split into the training, validation, and test subsets. The subsets other than the comparison set could have overlapping data points between folds (conditions).

Figure 8.8 depicts the K-fold comparison set split procedure. The data distributed in the folds, then will be fed into the pre-processing pipeline before using them to develop models.

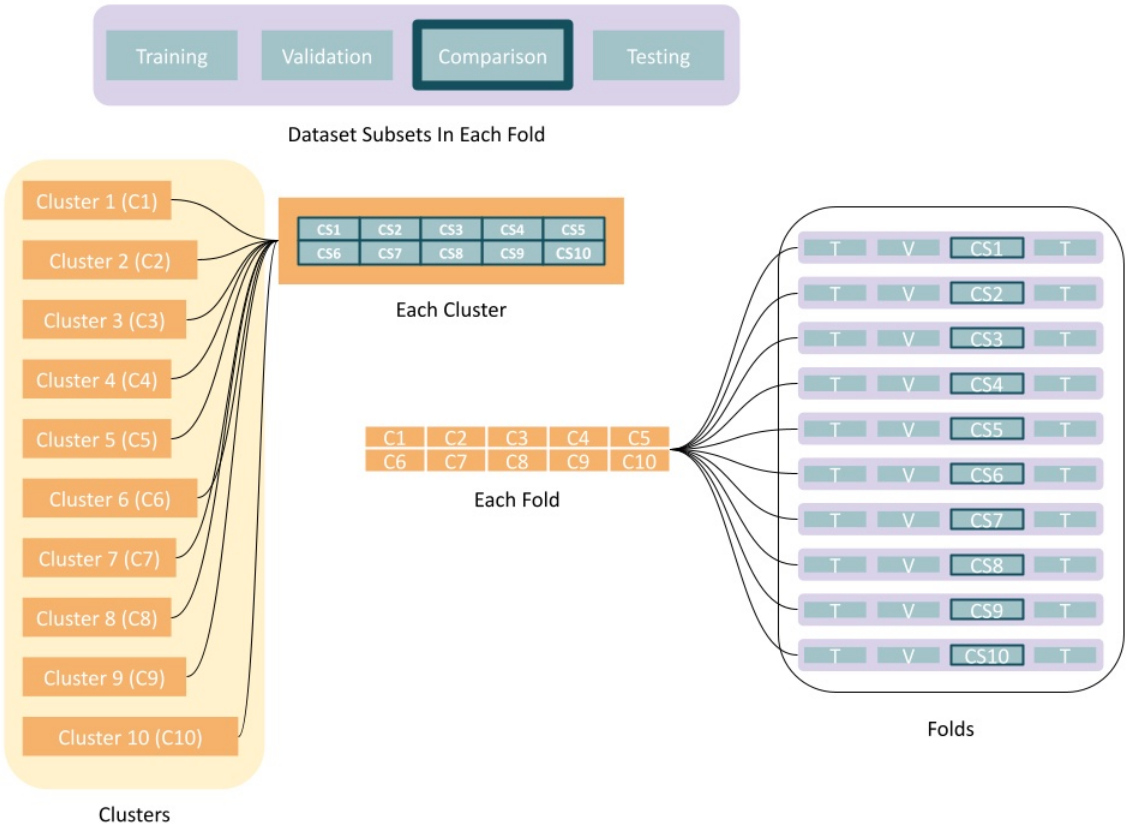


Figure 8.8: K-Fold Comparison Set Dataset Splitting.

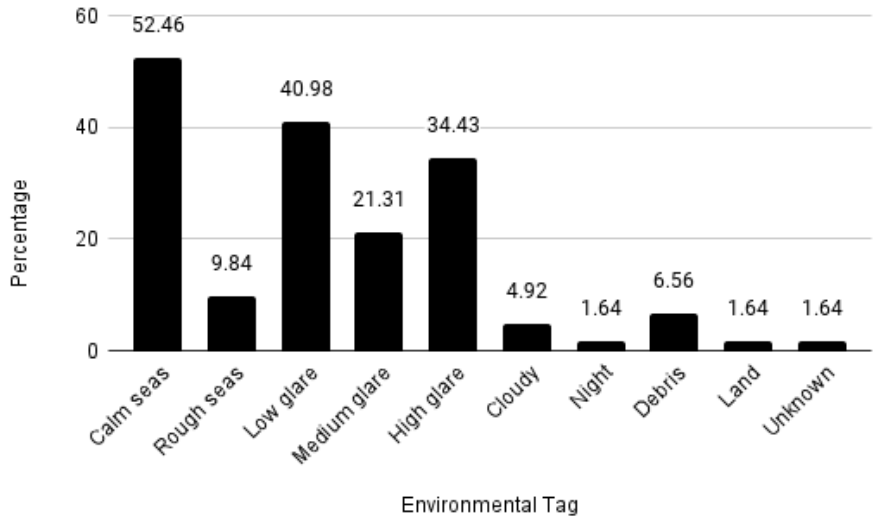


Figure 8.9: Environmental Tags Distribution of Comparison Data of Condition 1

8.4 Model Training and Evaluation (Experiment)

In this section, we discuss three conditions of manually labeling, labeling with the aid of an object detection model, and labeling with the assistance of a semantic segmentation model. We described the numeric metric used in this experiment in Appendix E and summarized the time measurements of these conditions in Table 8.5.

8.4.1 Condition 1: Manually Labeling

Manual labeling is the first condition to measure the time of three phases of labeling. This is the control condition for the experiment.

Comparison Set Data

In this condition, the expert receives a batch of images. The received batch is the comparison set for this fold. The batch contains 61 images in total containing 28 images that are empty of any object of interest. All the images are randomly flipped horizontally and vertically. The expert is not aware of the number of valuable images in the batch. The batch has a total of 114 objects. Figure 8.9 illustrates the percentage of each environmental tag inside the comparison set.

Phases

Phase one begins with the process of going through images to find the images with objects in them. The labeler deletes empty images, and phase one ends when the expert has all the valuable images ready. Phase two is the process of uploading the images into the labeling software. The final phase starts at the beginning of annotating the objects in the software and ends with exporting the annotations.

Time Measurement

For this condition, the first phase took 93 minutes. The second phase took 6 minutes. Phase 3 for this condition took 41 minutes, totaling 140 minutes for this condition.

8.4.2 Condition 2: Object Detection Model

In the second condition, we want to observe the effect of an object detection model on the amount of time needed for annotating images. Our object detection model is Faster R-CNN that we discussed in Chapter 6.

Model Training

For this condition, we created the dataset from the original images. The original images have a spatial resolution of 4912×7360 that is not suitable for Faster R-CNN. We cropped the images so that we described in Chapter 5 on Pre-Processing with both the grid and random cropping methods. The crop size for this model is 800×1200, preserving the original aspect ratio. All of the object annotations for object detection are bounding boxes. We used bounding box annotations and bounding boxes extracted from elliptic labeling.

To the best of our knowledge, there is no publicly available dataset similar to our dataset in appearance, resolution, and annotations. Hence, similar to the related works [74], we do transfer learning in the form of pre-training the networks with `COCO 2017` dataset. We fine-tune the pre-trained network on our dataset. We fine-tuned all the layers since fine-tuning all of the network weights for the new dataset yields better performance than freezing the feature extractors [98].

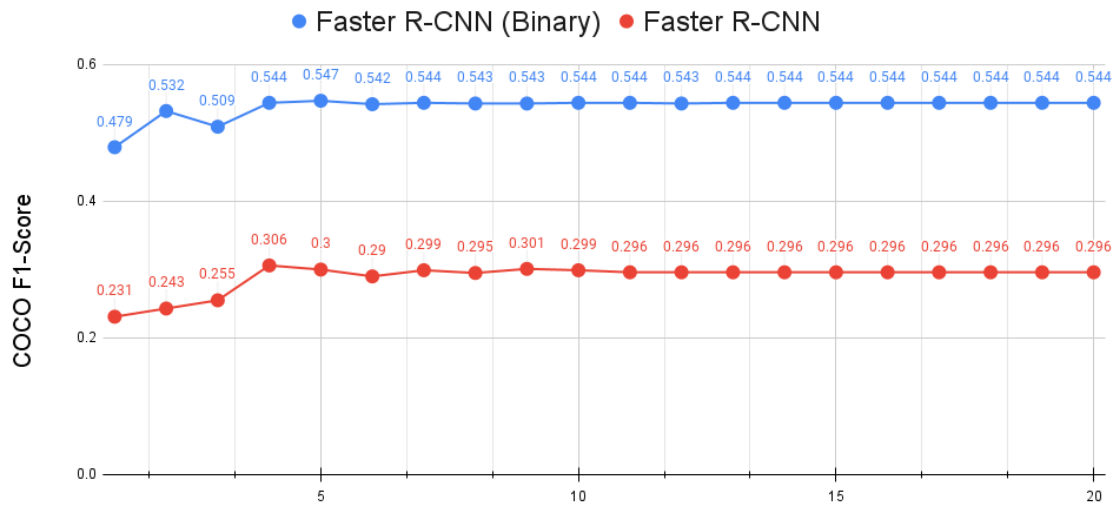


Figure 8.10: Comparison between COCO F1-Score of Faster R-CNN on the validation set when trained on the complete class list and binary set of classes.

Referring to some of our earlier model trainings in this project, we found that a Faster R-CNN model performs considerably better when the task is defined based on binary classes of background vs. object. Figure 8.10 shows a curve of COCO F1-Score of Faster R-CNN. We compute the COCO F1-Score based on Equation 8.1 with COCO Precision and Recall (see Appendix E for definitions of these metrics). For bootstrapping the data labeling process, localization is more important than the identification of the object. Hence, our model had two classes of background and object.

$$\text{COCO } F_1\text{-Score} = 2 \times \frac{\text{COCO Precision} \times \text{COCO Recall}}{\text{COCO Precision} + \text{COCO Recall}} \tag{8.1}$$

To fine-tune the architecture, we started by changing the Faster R-CNN head for our purpose of background vs. object detection. We fine-tuned the network for 30 epochs. We used Stochastic Gradient Descent (**SGD**) with momentum and weight decay as the optimizer to optimize our network with mini-batch size of 8. We also have a step learning rate scheduler. The initial learning rate of the fine-tuning is $5e-3$, and the optimizer's momentum and weight decay parameters are 0.9 and $5e-4$, respectively. The step learning rate scheduler multiplies the learning rate by 0.1 in every three epochs. Figure 8.11 depicts the PASCAL VOC metric of the model during the fine-tuning procedure. Figure 8.12 shows the complete COCO metrics of

this model at the end of epoch 30 of the fine-tuning.

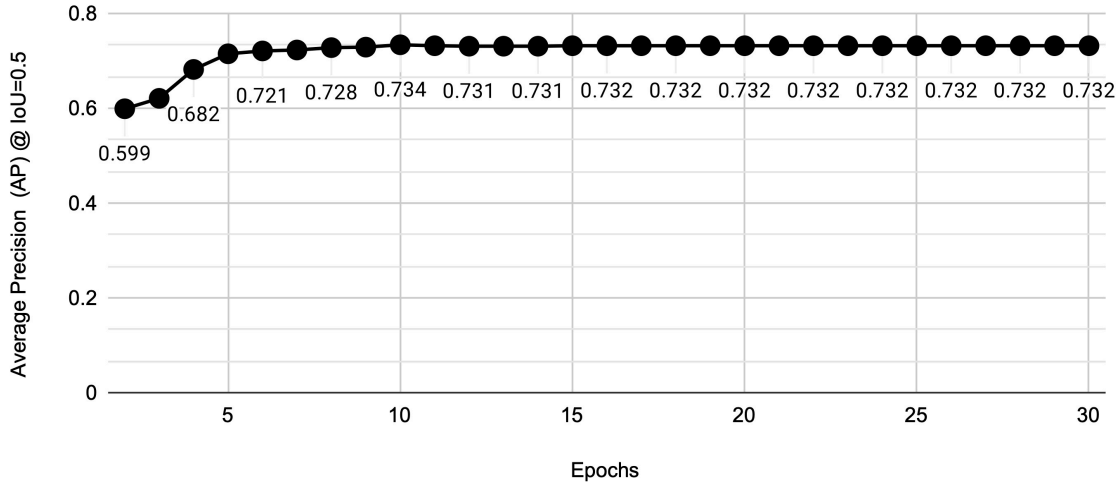


Figure 8.11: PASCAL VOC metric of the object detection. The alternate data points are annotated.

Average Precision	@[IoU=0.50:0.95	|area=all	|maxDets=100] =	0.404
Average Precision	@[IoU=0.50	|area=all	|maxDets=100]=	0.732
Average Precision	@[IoU=0.75	|area=all	|maxDets=100]=	0.413
Average Precision	@[IoU=0.50:0.95	|area=small	|maxDets=100]=	0.376
Average Precision	@[IoU=0.50:0.95	|area=medium	|maxDets=100]=	0.502
Average Precision	@[IoU=0.50:0.95	|area=large	|maxDets=100]=	0.356
Average Recall	@[IoU=0.50:0.95	|area=all	|maxDets= 1]=	0.396
Average Recall	@[IoU=0.50:0.95	|area=all	|maxDets= 10]=	0.550
Average Recall	@[IoU=0.50:0.95	|area=all	|maxDets=100]=	0.555
Average Recall	@[IoU=0.50:0.95	|area=small	|maxDets=100]=	0.532
Average Recall	@[IoU=0.50:0.95	|area=medium	|maxDets=100]=	0.638
Average Recall	@[IoU=0.50:0.95	|area=large	|maxDets=100]=	0.507

Figure 8.12: COCO metrics on the last epoch on the validation set

Comparison Set Data

We did an inference step with the confidence threshold of 0.5 to get model predictions on the comparison dataset using the fine-tuned model. The labeler receives two copies of the comparison set. In one copy, the images are annotated with model predictions, and the other one contains clean images. The comparison set contains 61 images in total containing 28 images that are empty of any object of interest. All the images are

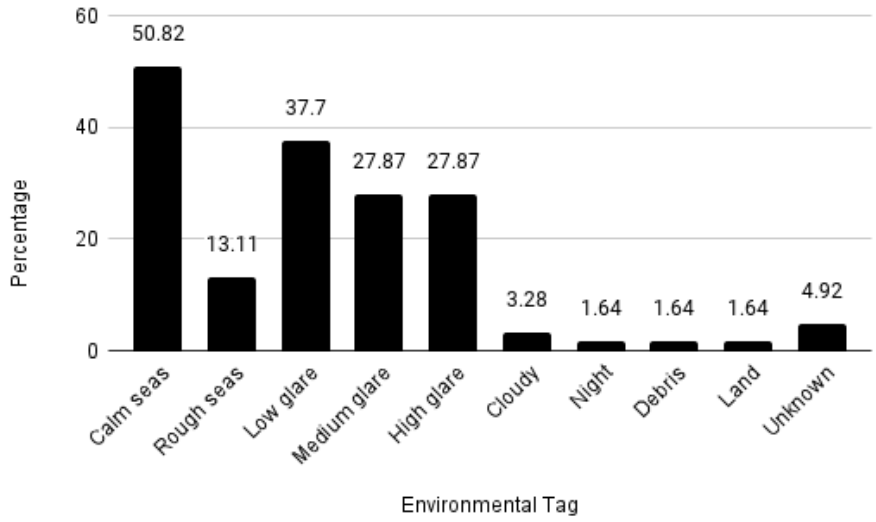


Figure 8.13: Environmental Tags Distribution of Comparison Data of Condition 2

randomly flipped horizontally and vertically. The expert is not aware of the number of valuable images in the batch. The batch has a total of 92 objects. Figure 8.13 illustrates the percentage of each environmental tag inside the comparison set.

Phases

In the first phase, the labeler goes through the annotated images to discover the images containing any objects of interest. The labeler deletes empty images, and phase one ends when the expert has all the valuable images ready. With the help of a provided script, the labeler uploads the soft annotations, predicted by the model, to the labeling software to modify them for the final label extractions. Phase two is the process of uploading the images and annotations into the labeling software. The final phase starts at the beginning of annotating the objects in the labeling software. The soft annotations show the location of the objects that the model found. If a predicted box does not contain an object (false positive), the labeler ignores it. On the contrary, if the box contains an object (true positive), the labeler verifies by changing its label, which is a class agnostic label, to the correct class label. Finally, if there are any objects without a bounding box around them (false negatives) or the bounding box does not contain the whole object, the labeler labels the object in a manual labeling fashion. Phase three ends with the exportation of the labels.

Model Evaluation

The model's performance on the comparison set, based on COCO metrics, is shown in Figure 8.14. Moreover, we counted the number of false positives and the number of false negatives in each confidence threshold from 0 to 100 percent with the step size of five percent (Figure 8.15). We set the threshold to 0.5 before the inference step so that the model has 973 false positives and seven false negatives in its predictions on the comparison set.

Average Precision	@[IoU=0.50:0.95	|area=all	|maxDets=100] =	0.384
Average Precision	@[IoU=0.50	|area=all	|maxDets=100]=	0.725
Average Precision	@[IoU=0.75	|area=all	|maxDets=100]=	0.359
Average Precision	@[IoU=0.50:0.95	|area=small	|maxDets=100]=	0.405
Average Precision	@[IoU=0.50:0.95	|area=medium	|maxDets=100]=	0.428
Average Precision	@[IoU=0.50:0.95	|area=large	|maxDets=100]=	0.194
Average Recall	@[IoU=0.50:0.95	|area=all	|maxDets= 1]=	0.318
Average Recall	@[IoU=0.50:0.95	|area=all	|maxDets= 10]=	0.526
Average Recall	@[IoU=0.50:0.95	|area=all	|maxDets=100]=	0.537
Average Recall	@[IoU=0.50:0.95	|area=small	|maxDets=100]=	0.544
Average Recall	@[IoU=0.50:0.95	|area=medium	|maxDets=100]=	0.593
Average Recall	@[IoU=0.50:0.95	|area=large	|maxDets=100]=	0.333

Figure 8.14: COCO metrics on the comparison set

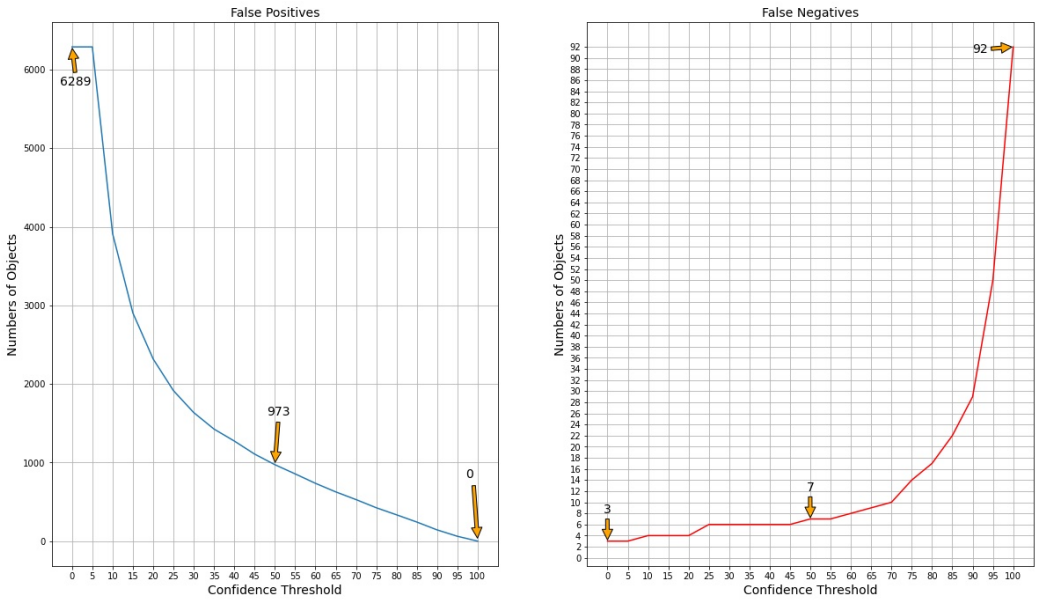


Figure 8.15: The number of false positives/negatives based on the confidence threshold of Faster R-CNN

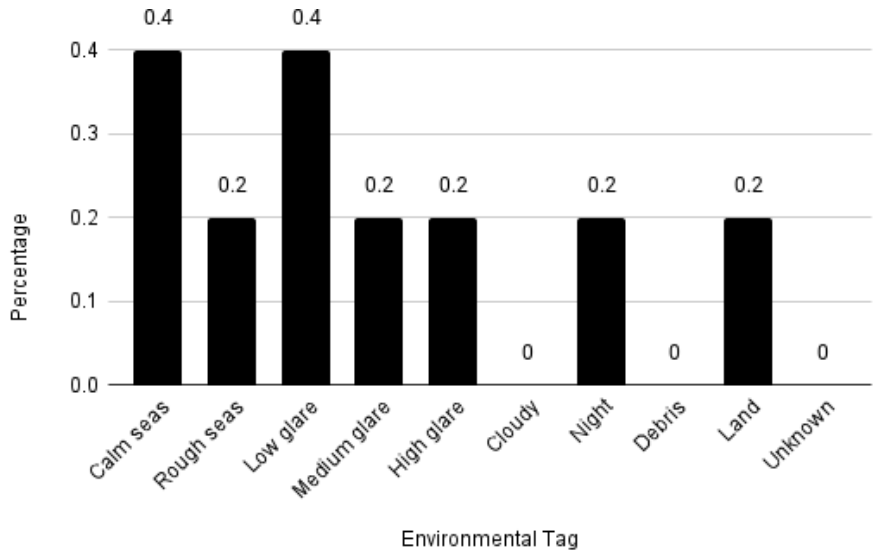


Figure 8.16: Environmental Tags of the images with false negatives in condition 2

Table 8.1: The number of false negatives for objects present in the comparison set in condition 2

Object Type	**Number of Objects**	**False Negatives**
Animal	39	2
Artifact	4	2
Object	2	0
Fishing Gear	37	2
Fishing Line	10	1

Table 8.1 has the list of 92 images in the comparison and the number of false negatives in each of them. The model successfully detected 37 instances of animals. Figure 8.16 is the chart for the environmental tags of 5 images in the comparison set that the model has false negative predictions. The false positives are not distributed among all the images evenly. There are four images with no false positives, and there is one image with 161 false positives. In Figure 8.17, we plot the environmental tag percentage of images with 20 or more false positives. There are 15 images with 20 or more false positives with 668 false positives in total, comprising approximately 69% of the total false positives on this condition. Figure 8.17 shows that glare amount is the dominant factor that causes false positive prediction. Table 8.2 has the breakdown of the false positives and false negatives per environmental tag.

Table 8.2: Summary of False positives and False negatives per environmental tags in condition 2. The first column is the list of environmental tags. The `Unknown sea state` and `Unknown glare amount` tags represent the images that did not have any sea state tags and glare amount tags, respectively. (x, y) tuples represent the x number of false positive/negative predictions that the model admitted in y images. For example, the model produced 410 false positive predictions for 28 images with the `Calm seas` tag, leaving three images without any false positives.

	(False Positives, # of images)	**(False Negatives, # of images)**	**Total # of images with this tag**
`Calm seas`	(410, 28)	(3, 2)	31
`Rough seas`	(189, 8)	(1, 1)	8
`Unknown sea state`	(374, 21)	(3, 2)	22
`Low Glare`	(204, 20)	(2, 2)	23
`Medium Glare`	(386, 17)	(2, 1)	17
`High Glare`	(287, 16)	(2, 1)	17
`Unknown glare amount`	(96, 4)	(1, 1)	4
`Cloudy`	(12, 2)	(0, 0)	2
`Night`	(29, 1)	(1, 1)	1
`Land`	(0, 0)	(2, 1)	1
`Debris`	(1, 1)	(0, 0)	1
`Unknown`	(67, 3)	(0, 0)	3

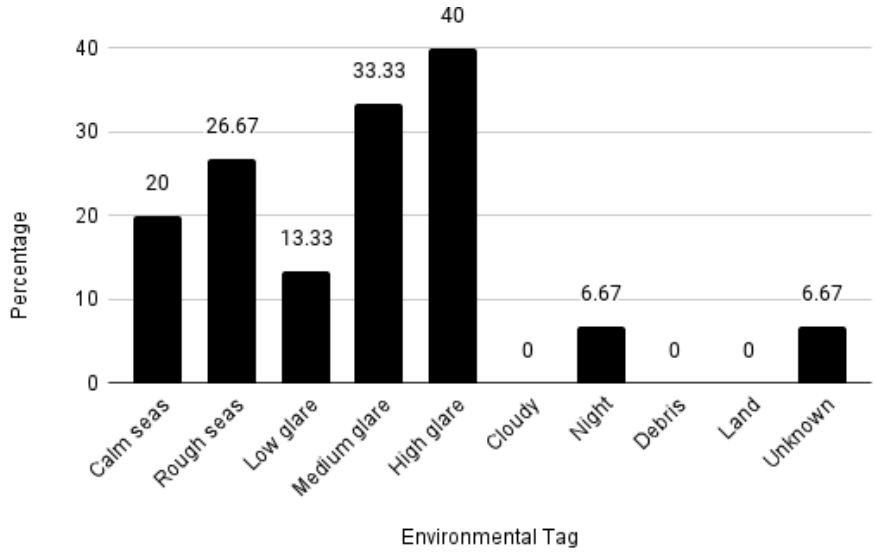


Figure 8.17: Environmental Tags of the images with 20 or more false positives in condition 2

Time Measurement

For this condition, the first phase took 59 minutes. The second phase took 6 minutes. Phase 3 for this condition took 48 minutes, totaling 113 minutes for this condition.

Qualitative Results

Our qualitative results suggest that the Faster R-CNN model can correctly detect objects in images with less adverse environmental factors such as sun glare with some false positives. However, the model produces overconfident false positives and false negative predictions in some samples with sun glare and reflections and other unfavorable environmental factors.

In Figure 8.18, there are two instances from `Animal` class. We set the detection threshold to 0.5 and run inference on these images using trained Faster R-CNN. The ground truth annotations have green boxes with green texts, and model predictions have orange boxes in the image. Since the model has only two classes (`Background`, `Object`), the label texts in the model predictions are not present. In this image, due to the high amount of sun glare, the model predicts numerous false positives.

Figure 8.19 is another example of our Faster R-CNN model. This image contains one `Turtle`. With the detection threshold of 0.5, the model correctly detects that

object. Although the sea is calmer and the amount of glare is lower than Figure 8.18, the model also predicts a few false positives.

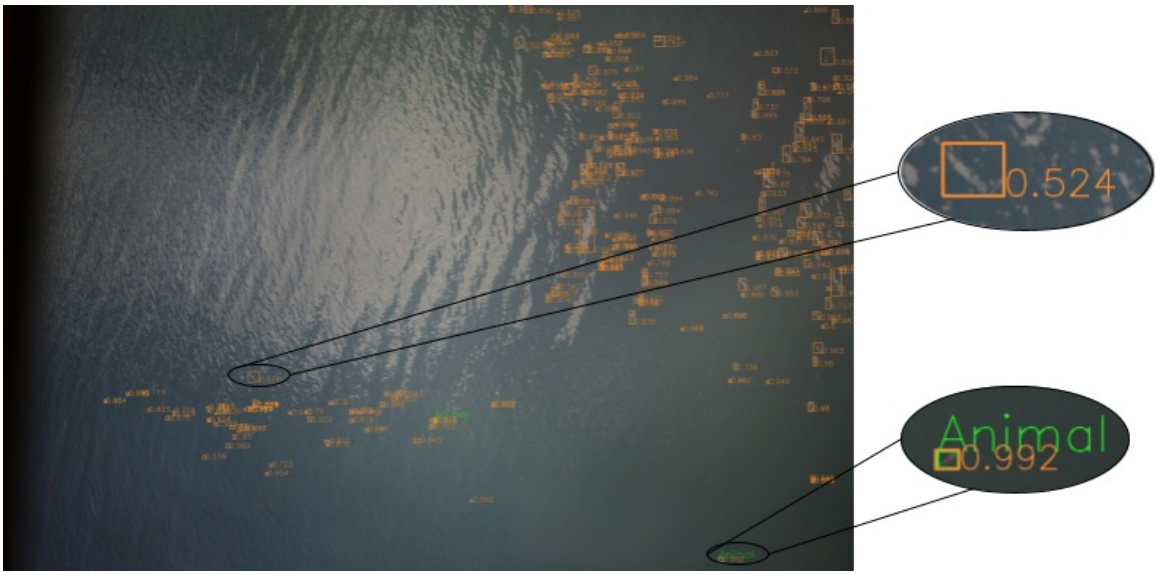


Figure 8.18: The image contains two animals. The model predicts both false positive and true positive results.

Figure 8.19: The image contains one `Turtle`. The model predicts both false positive and true positive results.

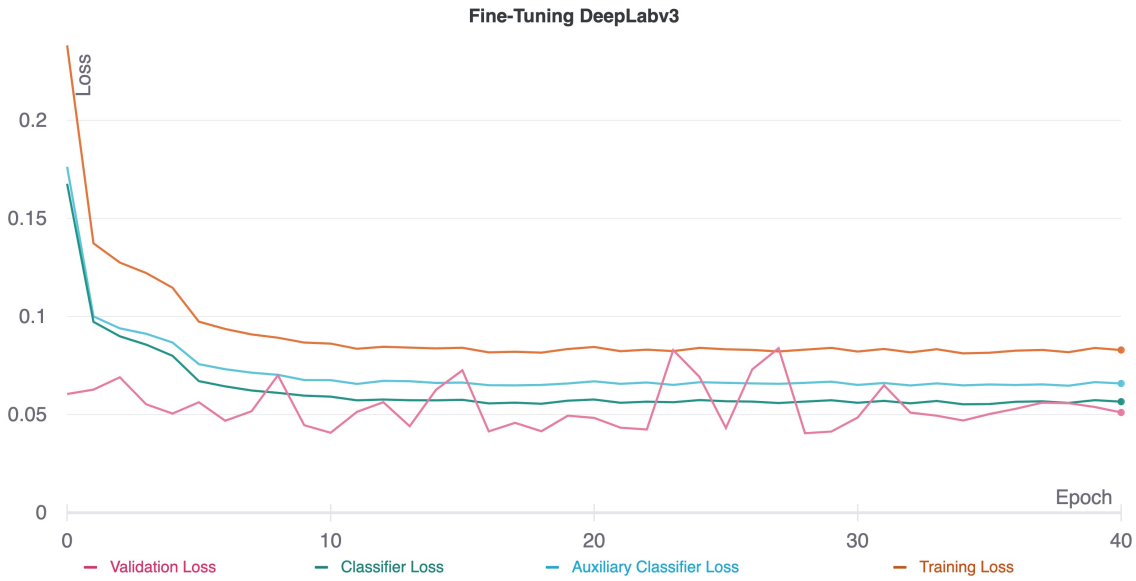


Figure 8.20: Loss Values for DeepLabv3 Fine-Tuning

8.4.3 Condition 3: Semantic Segmentation

In the third condition, we want to observe the effect of a semantic segmentation model on the amount of time needed for annotating images. We selected DeepLabv3 architecture discussed in Chapter 7 with ResNet101 backbone for semantic segmentation.

Model Training

Semantic Segmentation model needs input images and ground truth masks. We extracted rectangular masks for the objects from bounding box annotations and elliptic masks from elliptic annotations. We prepared the data using the extracted masks and original images by cropping them similar to the second condition with a crop size of 400×600. As mentioned before, there is no similar open data for pre-training and transfer learning to the best of our knowledge. Therefore, also for this model, we do the transfer learning from a pre-trained model with the source dataset of `COCO 2017`. `COCO 2017` has 20 object classes, including the background class, the final belief map has 21 channels. Our dataset has 19 classes, and considering the background, the number of channels in the belief map is 20. But, since we have only one sample from the `Humpback` whale, we do not use it for training, and we fine-tune the model for 19 channels in the belief map for 18 object classes and one background class. For the fine-tuning, we utilized an **auxiliary classifier** that is one of the successful ideas

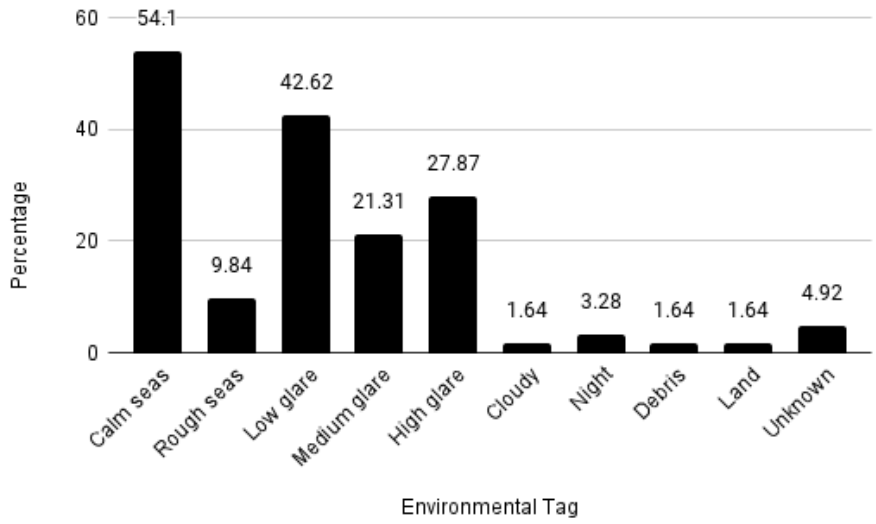


Figure 8.21: Environmental Tags Distribution of Comparison Data of Condition 3

in the literature. Auxiliary classifiers can boost gradient signals in the intermediate layers [40]. The auxiliary classifier is only being used during training, and for the inference, the primary decoder produces the predictions. For auxiliary classifier, we used a fully-convolutional classifier head, similar to the classifier head in the decoder part of the fully-convolutional network (FCN) with the weighted loss of 0.4. The FCN head gets the convolutional feature map from `conv4_x` (figure 7.5) of the backbone ResNet101. We set the weight of the primary segmentation head (DeepLabv3 head) to 1.0. The loss criterion for both of these heads was Cross-Entropy Loss. Figure 8.20 shows the loss of the network during the fine-tuning. Using **Adam** optimizer, we fine-tuned this network for 40 epochs with the mini-batch size of 4. The initial learning rate was $1e-4$. The learning rate gets multiplied by 0.1 in every five epochs by a step learning rate scheduler. Figure 8.20 shows the loss of the network during the fine-tuning.

Comparison Set Data

An inference step has been done to get model predictions on the comparison dataset using the fine-tuned model. For the purpose of having comparable predictions between conditions, we converted the prediction masks to bounding boxes using the method discussed in Appendix D. The labeler receives two copies of the comparison set.

In one copy, the images are annotated with model predictions, and the other one contains clean images. The comparison set contains 61 images in total containing 27 images that are empty of any object of interest. All the images are randomly flipped horizontally and vertically. The expert is not aware of the number of valuable images in the batch. The batch has a total of 81 objects. Figure 8.21 illustrates the percentage of each environmental tag inside the comparison set.

Phases

The first two phases are the same as the phases for the second condition, but the final phase differs in some details. If a prediction box contains an object (true positive), the labeler verifies the class and the location by changing its label to the correct class label. For this condition, the model predictions have the classes in addition to the location.

Model Evaluation

For this condition, the DeepLabv3 reaches the mIoU of 0.4142. Note that we used noisy ground truth masks and do not have the ideal set of ground truth labels with crisp edges of objects similar to benchmark datasets. We are aware that the computed metric is not as accurate as those in the benchmark dataset, and the noise in the masks affects computing metrics. The model produced 25 false negatives in 15 images. Table 8.3 has a list of classes and the number of false negatives. Figure 8.22 shows the distribution of environmental tags in these 15 images. The model produced 1065 false positive predictions. There are 21 images with 20 or more false positives with a max number of 84 for an image. Figure 8.23 depicts the distribution of environmental tags for these 21 images. There are 9 images that the model that did not produce any false positives. Table 8.4 has the breakdown of the false positives and false negatives per environmental tag.

8.4.4 Time Measurement

For this condition, the first phase took 41 minutes. The second phase initially took 10 minutes, but we created an optimized the script for this phase the time was reduced

Table 8.3: The number of false negatives for objects present in the comparison set in condition 3

Object Type	**Number of Objects**	**False Negative**
Animal	20	3
Shark	4	4
WNR	2	2
Minke	2	1
Artifact	2	1
Object	2	0
Fishing Gear	48	1
Fishing Line	1	0

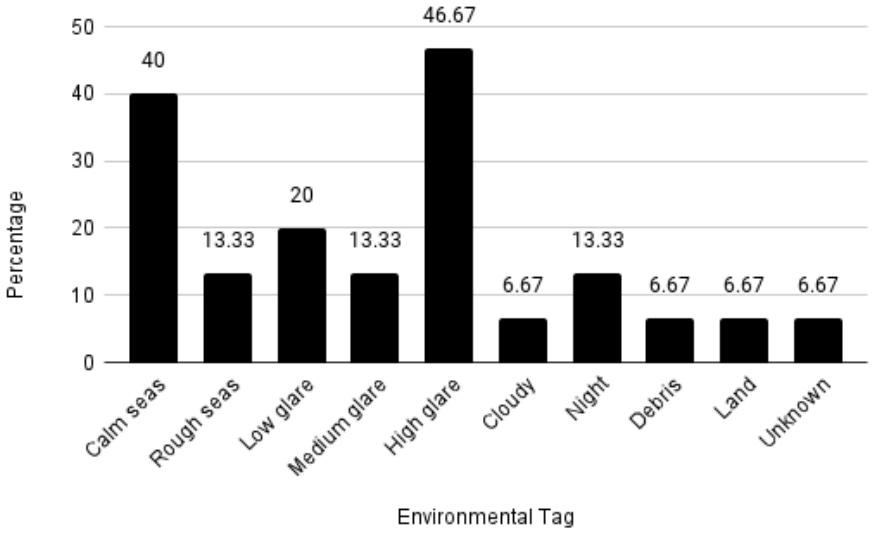


Figure 8.22: Environmental Tags of the images with false negatives

Table 8.4: Summary of False positives and False negatives per environmental tags in condition 3. The first column is the list of environmental tags. The `Unknown sea state` and `Unknown glare amount` tags represent the images that did not have any sea state tags and glare amount tags, respectively. (x, y) tuples represent the x number of false positive/negative predictions that the model admitted in y images. For example, the model produced 485 false positive predictions for 27 images with the `Calm seas` tag, leaving three images without any false positives.

	(False Positives, # of images)	**(False Negatives, # of images)**	**Total # of images with this tag**
`Calm seas`	(485, 27)	(10, 6)	33
`Rough seas`	(115, 5)	(2, 2)	6
`Unknown sea state`	(465, 20)	(13, 7)	22
`Low Glare`	(204, 20)	(3, 3)	26
`Medium Glare`	(274, 13)	(3, 2)	13
`High Glare`	(407, 16)	(11, 7)	17
`Unknown glare amount`	(16, 3)	(8, 3)	5
`Cloudy`	(36, 1)	(1, 1)	1
`Night`	(0, 0)	(7, 2)	2
`Land`	(30, 1)	(1, 1)	1
`Debris`	(83, 1)	(1, 1)	1
`Unknown`	(16, 3)	(1, 1)	3

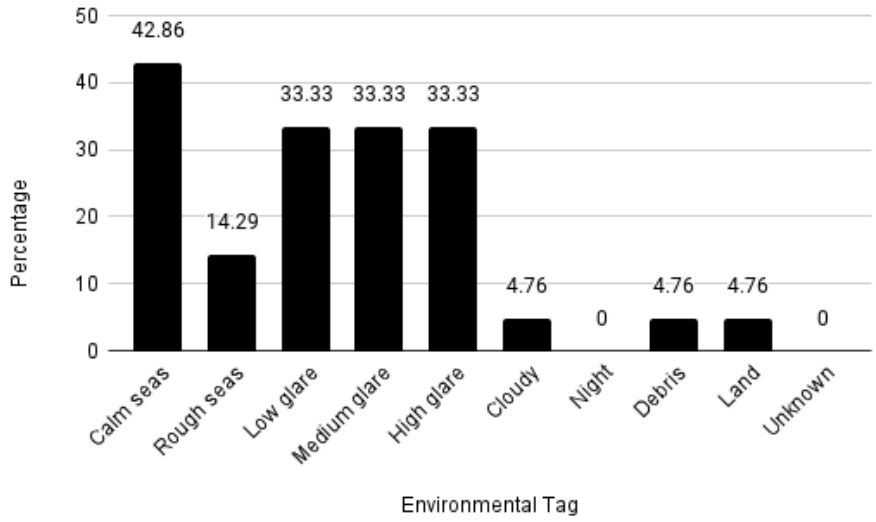


Figure 8.23: Environmental Tags of the images with 20 or more false positives

to 6 minutes. Phase three for this condition took 62 minutes, totaling 109 minutes for this condition.

Qualitative Results

Figure 8.24 is an example of the qualitative result of the semantic segmentation model. In this image, there is only one instance of `LBWNR` (as a reminder, LBWNR stands for "Large Baleen Whale not Right"). We depicted the elliptic ground truth mask of the object in green color and the model prediction's mask in orange. Ground truth and predictions have bounding boxes around the mask for better visibility with darker color shades than the overlaid segmentation mask. The model correctly located the animal but misclassified it. The model predicts an oval mask (not a perfect ellipse) containing the animal's tail, while the ellipse ground truth for the object does not include the tail.

Figure 8.25 is another example of the qualitative result of the semantic segmentation model. In this image, there is only one instance of `Humpback`. We depicted the ground truth in green color and model predictions in orange. The model missed the animal and predicted several other false positives. Since we only have one example of this class, we did not use this for training and the model has never seen any other object from this class. This example also supports the idea of using noisy annotations

for semantic segmentation. For example, the model predicted a false positive object (the zoomed-in false positive) that is a wave and has no ground truth mask; the prediction's segmentation mask (a light shade of orange inside the orange bounding box) has a wave shape that has never appeared in the training data.

Figure 8.26 contains one instance of `ULA` (ULA: Unidentified Large Animal). The model correctly localized the animal labeled it as `Sunfish` and predicted several other false positives. The prediction mask for this object with a light orange color shade also has a non-elliptic and non-rectangular shape. The prediction mask is more accurate rather than the rectangular ground truth mask.

The training data for semantic segmentation contains a mixture of rectangular masks (produced from bounding boxes) and elliptic masks. Hence, the model has never seen any polygon mask, and the masks are always noisy. We observed that the model tends to correctly identify the shape of the objects rather than perfect ellipses or rectangles. The qualitative results confirm that the assumption of using noisy annotation in the presence of unbalanced data (in favor of the water (background) class) is valid.

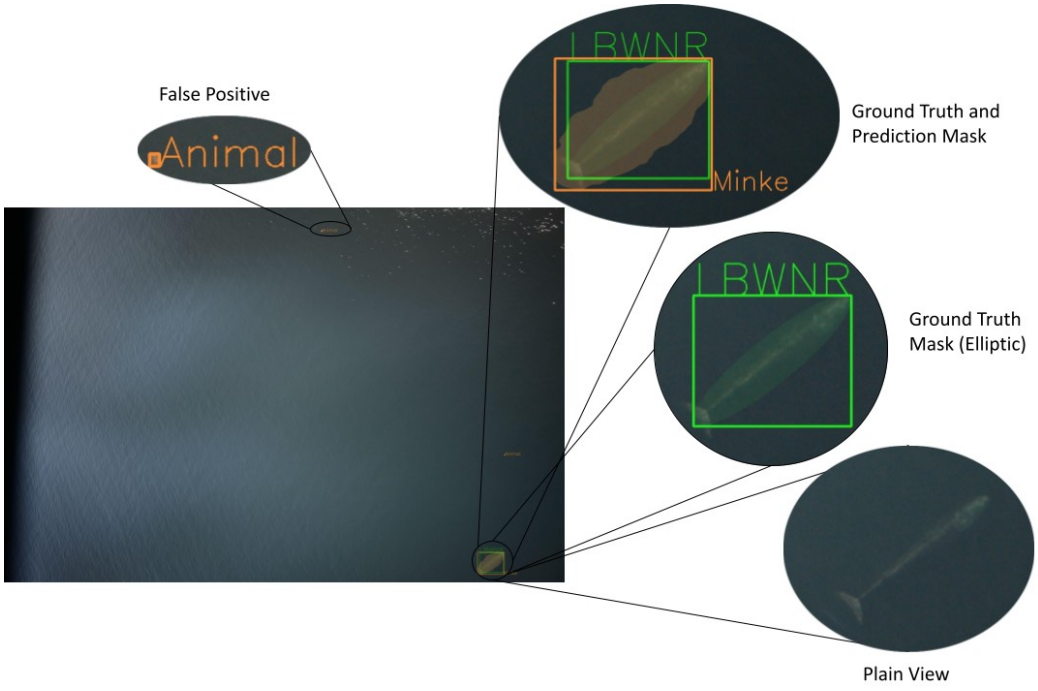


Figure 8.24: The image contains one instance from `LBWNR` class. The model predicts false positive and false negative results.

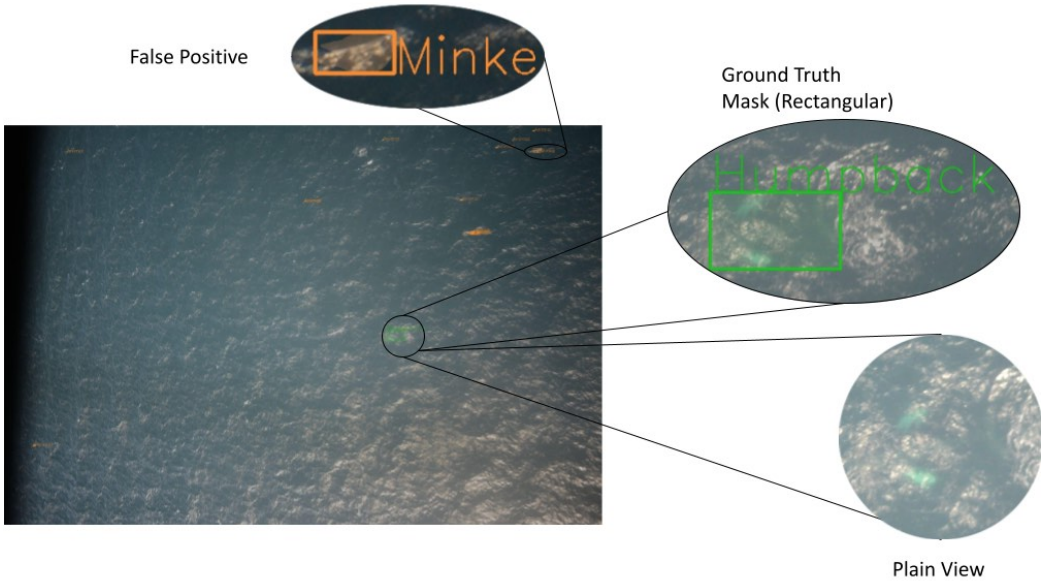


Figure 8.25: The image contains one instance from `Humpback` class. The model predicts both false positive and false negative results.

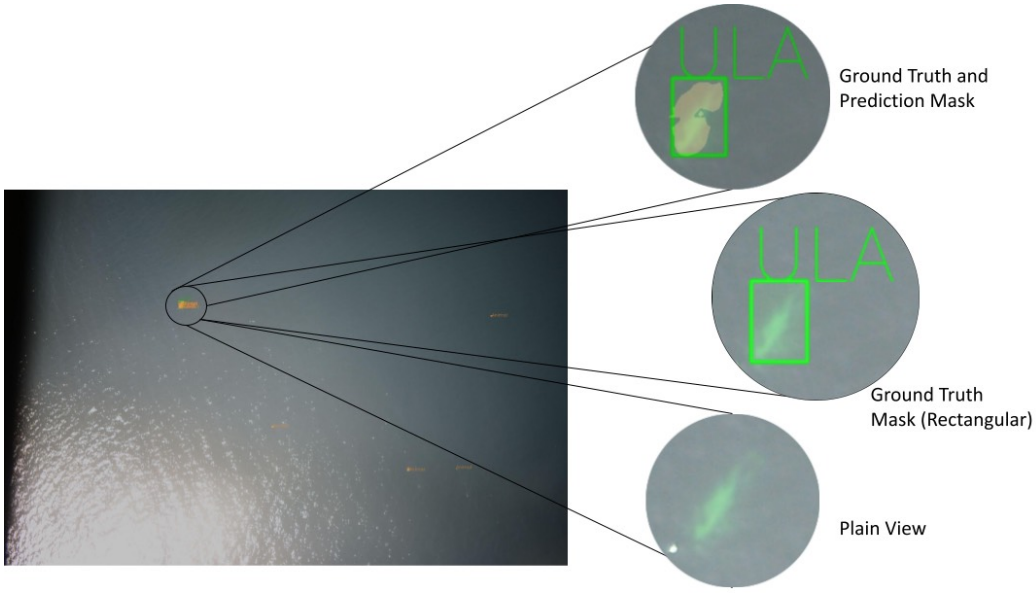


Figure 8.26: The image contains one instance from `ULA` class. The model predicts both false positive and true positive results.

Table 8.5: The time measurements and the number of false positives and false negatives in each condition

	Phase 1 Time (minutes)	Phase 2 Time (minutes)	Phase 3 Time (minutes)	Total Time (minutes)	Number of False Positives	Number of False Negatives
Manually Labeling	93	6	41	140	-	-
Object Detection	59	6	48	113	973	7
Semantic Segmentation	41	6	62	109	1065	25

8.4.5 Summary

Table 8.5 contains the time measurements and number of false positives and false negatives for each condition, except for the manually labeling condition that has no machine learning prediction and false positive and false negative counts. The time measurements and the number of false positives and false negatives suggest that a computer vision model helps to reduce the time needed for phase one of image annotation. The semantic segmentation model has nine images without false positives, while object detection has four images with no false positive predictions. Time measurements suggest that with approximately the same number of total false positives, the number of images without any false positives plays a dominant role in accelerating phase one. Phase one of the annotation workflow consumes the majority of the time. Finding images with objects of interest is indeed a more laborious task than annotating found objects in images. That is, these models are helpful in bootstrapping data collection. However, AI-in-the-loop is not an entire solution for an extremely under-resourced initial dataset. Several classes have a small number of examples, and some of them have only a few. A slightly larger dataset, especially with more examples in underrepresented classes, may be sufficient to bootstrap the training and collection with effectively no false negatives.

It is worthwhile to mention that due to an agreement with the data owner, we only have the right to include a limited number of data examples in the publication. Moreover, as mentioned before in the dataset section, the intention behind using

Figure 8.27: Figure 8.18 without Model Prediction

object class labels other than animals is not to detect them. We use them in the training set and model evaluation to make the models less sensitive to those objects and to evaluate the approaches, respectively.

8.4.6 Discussion

The results show that sun glare and rough sea state (e.g. more waves) are dominant in the images with a high false positive rate and false negative rate. Here we want to elucidate the reason behind this with an example photo. Figure 8.27 is an image with noticeable sun glare, and the model predicted several false positives on it (see Figure 8.18 for the predictions). Figure 8.28 is a zoomed crop of a region from Figure 8.27. We applied the Canny edge detection algorithm on this crop to find the edges with a minimum threshold of 5 and a maximum threshold of 50. As we can see in Figure 8.28, the edges of the animal are similar to the edges of the waves. This similarity makes object detection and semantic segmentation extremely challenging, resulting in high false positive and false negative rates.

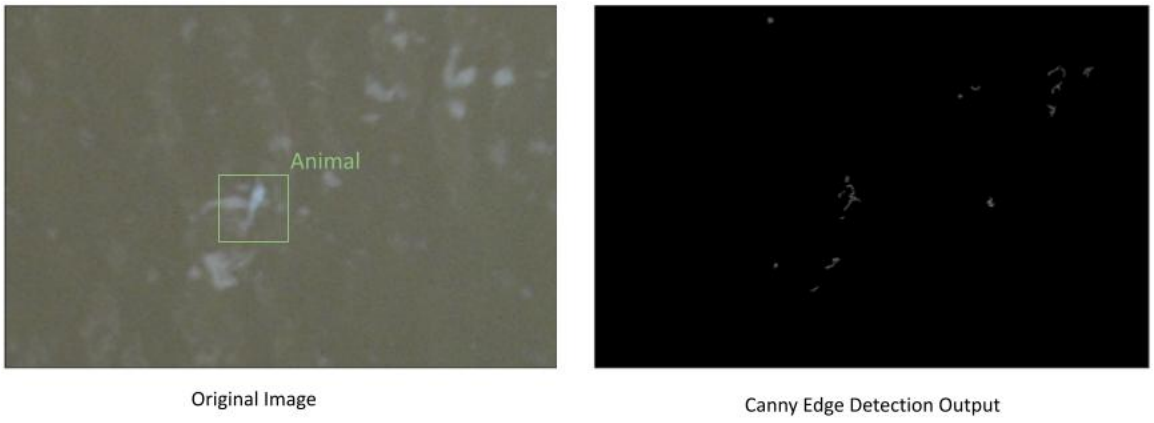


Figure 8.28: A crop from 8.27 and the output of the Canny Edge Detection algorithm on it.

Detecting animals in this image is a complex problem due to the small sizes of objects. Experts usually incorporate more context around the animals themselves to ensure they correctly detected them. For example, in the case of finding an animal-shaped object in the water, a tailing wave produced by the swimming of the animal is hard evidence to consider the object as an animal. This type of context originated from common knowledge or advanced studies on animals that are not included in the dataset with object annotations. However, more data can mitigate this challenge. For example, algorithms such as semantic segmentation incorporate visual context around a pixel to classify it.

Chapter 9

Conclusion and Future Work

In this chapter, we discuss the conclusion of this research and some possible future work tracks in this field.

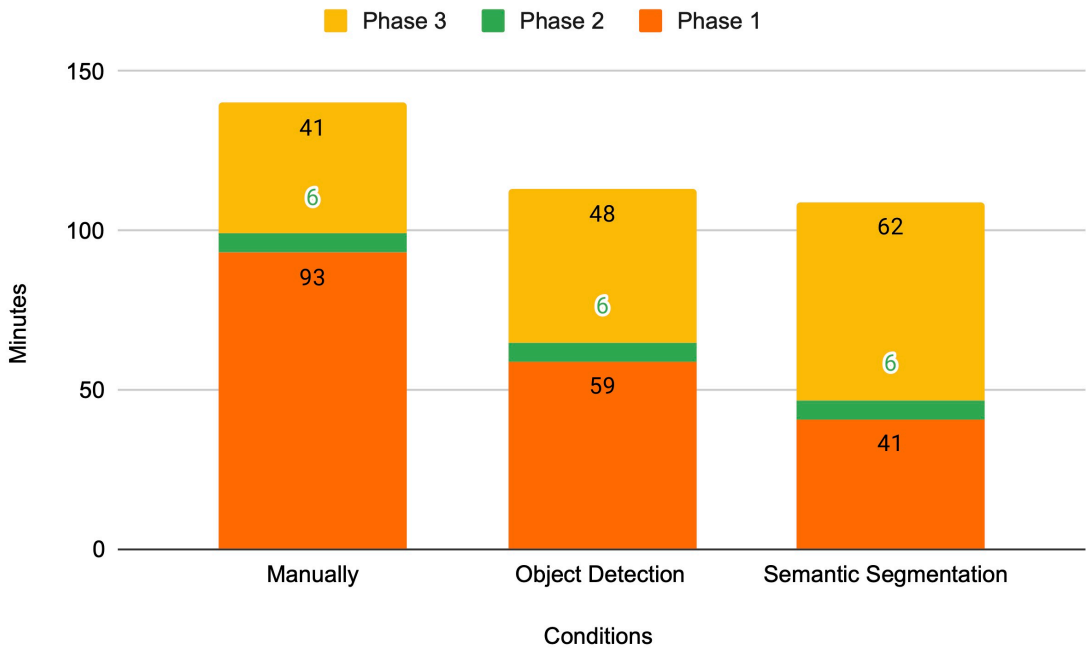
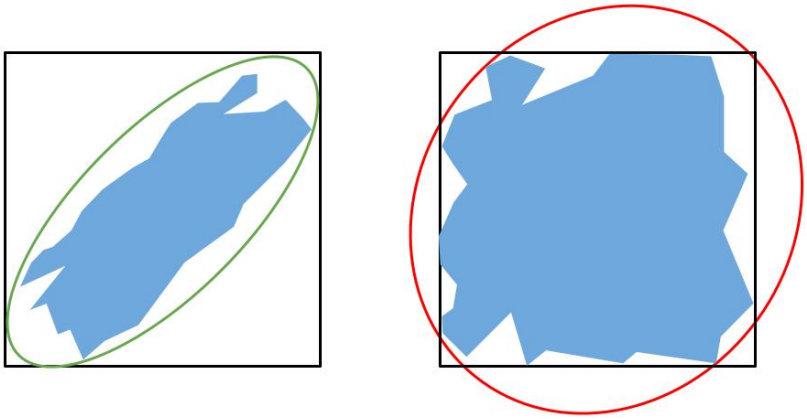


Figure 9.1: The time measurements for three conditions of labeling

9.1 Conclusion

In this work, we develop a computer vision system to help biologists detect endangered whales in an AI-in-the-loop data collection setup. Given access to limited aerial imagery data, we implemented two computer vision models for this purpose: object detection and semantic segmentation. We introduced elliptic object annotation for the semantic segmentation model that captures less noise than bounding box object



a b

Figure 9.2: Usage of elliptic labeling for two different type of applications.

annotation in the ground truth segmentation masks while constrained by the labeler's time.

Masks extracted from the elliptic labeling scheme captured 43% less noise in the ground truth mask than rectangular masks extracted from bounding boxes. Elliptic labeling assumes that the shapes of objects are near-oval in the images. For semantic segmentation applications that this assumption holds validity, elliptic labeling is better than bounding box labeling. Nevertheless, polygon labeling is the standard way of object annotation in semantic segmentation, and it is recommended over elliptic labeling if it is feasible to collect based on time or financial budget. To elucidate the difference between elliptic and bounding box labeling, we show two different shapes of the objects in Figure 9.2. In Figure 9.2 (a), the object has a near-oval shape that can be annotated effectively with less noise in the ground truth mask than bounding box annotation. On the other hand, Figure 9.2 (b) depicts another shape type that cannot capture elliptic labeling without adding more noise to the ground truth mask, so for this case, bounding box labeling is more suitable.

The data mainly comprises water pixels and forces data imbalance. We leveraged this imbalance by training the semantic segmentation architecture using noisy target masks produced by the bounding box and elliptic annotations. This approach was successful and qualitative results show the model's tendency to correctly recognize the shapes of the objects rather than the shapes of noisy labels present during training. The developed models reach PASCAL VOC metric of 73.2%, distinguishing objects from water, for object detection model and a mIoU of 41.42% on the dataset, for

semantic segmentation model.

The data scarcity and imbalance made the zero-false-negative rate infeasible. We minimized false negatives while having *few enough* false positives that it could still help an expert annotator accelerate the annotation process itself. Having a computer vision system as such enables a bootstrapping data collection process. This allows gradually building a set of increasingly accurate models, each of which in turn helps build larger datasets more efficiently and so on.

We evaluated the performance of the system for the downstream bootstrapping task with an AI-in-the-Loop experiment. We designed and conducted an experiment imitating expert user's object annotation workflow. The experiment's result (Figure 9.1) shows that using an AI-in-the-loop setup takes less time to collect the same amount of labeled data than manual data collection despite having a few false negative predictions. Despite the significant time-saving results, the proposed approach cannot entirely solve the challenge of the extremely small dataset. The results show the possibility of improvement with a slightly larger dataset that may be sufficient to bootstrap the training and collection with effectively no false negatives.

9.2 Future Work

During this work, we observed that high confidence and wrong predictions could be probable problematic outcomes. Neural networks produce high confidence results in the case of being fed with out-of-distribution (OOD) data. We believe that training dataset size can play a crucial role in the occurrence of this incident. For future work, a study of out-of-distribution data is a possible direction. Methods such as Gram-OOD [93] and Gram-OOD* [99] may help to produce reliable classification scores and yield better AI-in-the-loop data collecting pipelines.

Another track of future work can be exploring approaches that incorporate environmental conditions. Comparing the environmental tags with the clusters shaped using K-means and the extracted features of data samples suggests the possibility of developing more advanced techniques to incorporate environmental conditions in computer vision models.

As mentioned in the related works section, whale surveying using acoustic data is more explored than aerial imagery. One possible future work is to use acoustic

data and aerial imagery for marine animal detection. Having both kinds of acoustic and aerial imagery can be beneficial. For example, a negative prediction using the acoustic domain can correct a false positive prediction in aerial imagery.

Bootstrapping data collection is crucial for data-scarce problems. Once data collection reaches a point that trained models can detect objects with no false negatives, some machine learning techniques can accelerate the process further. Some recent advances in few-shot learning [100, 101] show the possibility of using them in the future that also can be applied in the context of whale detection in aerial imagery. However, these approaches are not applicable in the context of scarce data, and they are not as accurate as supervised methods. In few-shot learning, some of the novel classes need a few labeled examples, while the base classes need an abundance of annotated images for training. With an abundance of unlabeled images and more annotated data, another track of future works can be leveraging the concept of semi-supervised learning. Recent studies such as "End-to-End Semi-Supervised Object Detection with Soft Teacher" [102] can be investigated in the context of whale detection. Moreover, once the dataset size grows in size and the classification scores become reliable, active learning can be another track of future works to facilitate data collection with labeling more useful examples for training more accurate models.

Sun glare is a dominant factor in aerial imagery that adversely affects the performance of computer vision models. Therefore, modeling the noise on aerial imagery and denoising images before feeding the images into the detection models can be studied as future work. Recent approaches such as Noise Flow [103] show the possibility of having a noise model for developing better denoiser networks.

Motivated by the experiment of using the binarized class list in Section 8.4.2, one possible approach for future work is to use several hierarchies of classes. Such as combining two or more classes into one that holds validity from a scientific standpoint.

We proposed an AI-in-the-loop setup to collect data for creating a whale detection dataset. We conducted an experiment to see the effect of our system on the speed of annotation. Having a fine-grained timed variant of the experiment with details of every image and the time it took for annotation can be beneficial for devising better strategies. To make this experiment possible, an interesting future work can be developing a software platform that combines all the phases in one place and logs

every action of the user without adding distractions to the annotation workflow.

An ideal detection system can detect whales and marine mammals with no false negatives and only a few false positives. This means the model learned the representations needed by getting numerous training data. This can bring up the interpretability of the model, especially when there is a disagreement between a human expert and the model. Interpretable object detection and semantic segmentation [104, 105] can be beneficial to find the features that the model looks for in the context of detecting whales.

Bibliography

[1] Zhengxia Zou, Zhenwei Shi, Yuhong Guo, and Jieping Ye. Object detection in 20 years: A survey, 2019.

[2] OpenStreetMap contributors. Planet dump retrieved from https://planet.osm.org . https://www.openstreetmap.org, 2017.

[3] Javier Rey. Faster r-cnn: Down the rabbit hole of modern object detection, Jan 2018.

[4] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection, 2017.

[5] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2016.

[6] Christian S Perone, Evan Calabrese, and Julien Cohen-Adad. Spinal cord gray matter segmentation using deep dilated convolutions. *Scientific reports*, 8(1):1–13, 2018.

[7] Ximin Cui, Ke Zheng, Lianru Gao, Dong Yang, and Jinchang Ren. Multiscale spatial-spectral convolutional network with image-based framework for hyperspectral imagery classification. *Remote Sensing*, 11:2220, 09 2019.

[8] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs, 2017.

[9] Alexandru Telea. An image inpainting technique based on the fast marching method. *Journal of graphics tools*, 9(1):23–34, 2004.

[10] J.G. Cooke. Eubalaena glacialis (errata version published in 2020). 2020. https://dx.doi.org/10.2305/IUCN.UK.2020-2.RLTS.T41712A178589687.en.

[11] Olga Koubrak, David L. VanderZwaag, and Boris Worm. Saving the north atlantic right whale in a changing ocean: Gauging scientific and law and policy responses. *Ocean & Coastal Management*, 200:105109, 2021.

[12] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, pages I–I, 2001.

[13] Paul Viola and Michael J Jones. Robust real-time face detection. *International journal of computer vision*, 57(2):137–154, 2004.

[14] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 886–893. Ieee, 2005.

[15] Ross Brook Girshick. *From rigid templates to grammars: Object detection with structured models*. The University of Chicago, 2012.

[16] Ross Girshick, Pedro Felzenszwalb, and David McAllester. Object detection with grammar models. *Advances in Neural Information Processing Systems*, 24:442–450, 2011.

[17] Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *IEEE transactions on pattern analysis and machine intelligence*, 32(9):1627–1645, 2009.

[18] Pedro F Felzenszwalb, Ross B Girshick, and David McAllester. Cascade object detection with deformable part models. In *2010 IEEE Computer society conference on computer vision and pattern recognition*, pages 2241–2248. Ieee, 2010.

[19] Ross Girshick, Forrest Iandola, Trevor Darrell, and Jitendra Malik. Deformable part models are convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[20] A Krizhevsky, I Sutskever, and GE Hinton. 2012 alexnet, 2012.

[21] Yann LeCun et al. Lenet-5, convolutional neural networks. *URL: http://yann. lecun. com/exdb/lenet*, 20(5):14, 2015.

[22] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation, 2014.

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *Lecture Notes in Computer Science*, page 346–361, 2014.

[24] Ross Girshick. Fast r-cnn. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448, 2015.

[25] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2016.

[26] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger, 2016.

[27] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement, 2018.

[28] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection, 2020.

[29] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. *Lecture Notes in Computer Science*, page 21–37, 2016.

[30] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection, 2018.

[31] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers, 2020.

[32] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable detr: Deformable transformers for end-to-end object detection, 2021.

[33] Ting Chen, Saurabh Saxena, Lala Li, David J. Fleet, and Geoffrey Hinton. Pix2seq: A language modeling framework for object detection, 2021.

[34] Shervin Minaee, Yuri Y. Boykov, Fatih Porikli, Antonio J Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. Image segmentation using deep learning: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2021.

[35] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[36] Alexander Kirillov, Kaiming He, Ross Girshick, Carsten Rother, and Piotr Dollár. Panoptic segmentation, 2019.

[37] Bowen Cheng, Maxwell D. Collins, Yukun Zhu, Ting Liu, Thomas S. Huang, Hartwig Adam, and Liang-Chieh Chen. Panoptic-deeplab: A simple, strong, and fast baseline for bottom-up panoptic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

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

[39] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

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

[41] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. In *Proceedings of the IEEE international conference on computer vision*, pages 1520–1528, 2015.

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

[43] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[44] Matthias Holschneider, Richard Kronland-Martinet, Jean Morlet, and Ph Tchamitchian. A real-time algorithm for signal analysis with the help of the wavelet transform. In *Wavelets*, pages 286–297. Springer, 1990.

[45] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs, 2016.

[46] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation, 2017.

[47] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation, 2018.

[48] Bashar Mohammad and Ronald McHugh. Automatic detection and characterization of dispersive north atlantic right whale upcalls recorded in a shallow-water environment using a region-based active contour model. *IEEE Journal of Oceanic Engineering*, 36(3):431–440, 2011.

[49] R. P. Morrissey, S. Jarvis, N. DiMarzio, J. Ward, and D. J. Moretti. North atlantic right whale (eubalaena glacialis) detection amp; localization in the bay of fundy using widely spaced, bottom mounted sensors. In *OCEANS 2006*, pages 1–6, 2006.

[50] I. Urazghildiiev, C.W. Clark, and T. Krein. Acoustic detection and recognition of fin whale and north atlantic right whale sounds. In *2008 New Trends for Environmental Monitoring Using Passive Systems*, pages 1–6, 2008.

[51] A.S.M. Vanderlaan, A.E. Hay, and C.T. Taggart. Characterization of north atlantic right-whale (eubalaena glacialis) sounds in the bay of fundy. *IEEE Journal of Oceanic Engineering*, 28(2):164–173, 2003.

[52] Kathleen M. Stafford, Christopher G. Fox, and David S. Clark. Long-range acoustic detection and localization of blue whale calls in the northeast pacific ocean. *The Journal of the Acoustical Society of America*, 104(6):3616–3625, 1998.

[53] Holger Klinck, David K. Mellinger, Karolin Klinck, Neil M. Bogue, James C. Luby, William A. Jump, Geoffrey B. Shilling, Trina Litchendorf, Angela S. Wood, Gregory S. Schorr, and Robin W. Baird. Near-real-time acoustic monitoring of beaked whales and other cetaceans using a seaglider™. *PLOS ONE*, 7(5):1–8, 05 2012.

[54] Katie A. Kowarski, Briand J. Gaudet, Arthur J. Cole, Emily E Maxner, Stephen P Turner, S. Bruce Martin, Hansen D. Johnson, and John E. Moloney. Near real-time marine mammal monitoring from gliders: Practical challenges, system development, and management implications. *The Journal of the Acoustical Society of America*, 148(3):1215–1230, 2020.

[55] Cédric Gervaise, Yvan Simard, Florian Aulanier, and Nathalie Roy. Optimizing passive acoustic systems for marine mammal detection and localization: Application to real-time monitoring north atlantic right whales in gulf of st. lawrence. *Applied Acoustics*, 178:107949, 2021.

[56] Mark F. Baumgartner, Julianne Bonnell, Peter J. Corkeron, Sofie M. Van Parijs, Cara Hotchkin, Ben A. Hodges, Jacqueline Bort Thornton, Bryan L. Mensi, and Scott M. Bruner. Slocum gliders provide accurate near real-time estimates of baleen whale presence from human-reviewed passive acoustic detection information. *Frontiers in Marine Science*, 7:100, 2020.

[57] Ming Zhong, Manuel Castellote, Rahul Dodhia, Juan Lavista Ferres, Mandy Keogh, and Arial Brewer. Beluga whale acoustic signal classification using deep learning neural network models. *The Journal of the Acoustical Society of America*, 147(3):1834–1841, 2020.

[58] Mahdi Esfahanian, Nurgun Erdol, Edmund Gerstein, and Hanqi Zhuang. Two-stage detection of north atlantic right whale upcalls using local binary patterns and machine learning algorithms. *Applied Acoustics*, 120:158–166, 2017.

[59] Hamed Mohebbi-Kalkhoran, Chenyang Zhu, Matthew Schinault, and Purnima Ratilal. Classifying humpback whale calls to song and non-song vocalizations using bag of words descriptor on acoustic data. In *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, pages 865–870, 2019.

[60] Xanadu C. Halkias, Sébastien Paris, and Hervé Glotin. Classification of mysticete sounds using machine learning techniques. *The Journal of the Acoustical Society of America*, 134(5):3496–3505, 2013.

[61] Peter J. Dugan, Aaron N. Rice, Ildar R. Urazghildiiev, and Christopher W. Clark. North atlantic right whale acoustic signal processing: Part i. comparison of machine learning recognition algorithms. In *2010 IEEE Long Island Systems, Applications and Technology Conference*, pages 1–6, 2010.

[62] Peter J. Dugan, Aaron N. Rice, Ildar R. Urazghildiiev, and Christopher W. Clark. North atlantic right whale acoustic signal processing: Part ii. improved decision architecture for auto-detection using multi-classifier combination methodology. In *2010 IEEE Long Island Systems, Applications and Technology Conference*, pages 1–6, 2010.

[63] Ali K. Ibrahim, Hanqi Zhuang, Nurgun Erdol, and Ali Muhamed Ali. A new approach for north atlantic right whale upcall detection. In *2016 International Symposium on Computer, Consumer and Control (IS3C)*, pages 260–263, 2016.

[64] Ho Chun Huang, John Joseph, Ming Jer Huang, and Tetyana Margolina. Automated detection and identification of blue and fin whale foraging calls by combining pattern recognition and machine learning techniques. In *OCEANS 2016 MTS/IEEE Monterey*, pages 1–7, 2016.

[65] Ann N. Allen, Matt Harvey, Lauren Harrell, Aren Jansen, Karlina P. Merkens, Carrie C. Wall, Julie Cattiau, and Erin M. Oleson. A convolutional neural network for automated detection of humpback whale song in a diverse, long-term passive acoustic dataset. *Frontiers in Marine Science*, 8:165, 2021.

[66] Jia jia Jiang, Ling ran Bu, Fa jie Duan, Xian quan Wang, Wei Liu, Zhong bo Sun, and Chun yue Li. Whistle detection and classification for whales based on convolutional neural networks. *Applied Acoustics*, 150:169–178, 2019.

[67] Peter C Bermant, Michael M Bronstein, Robert J Wood, Shane Gero, and David F Gruber. Deep machine learning techniques for the detection and classification of sperm whale bioacoustics. *Scientific reports*, 9(1):1–10, 2019.

[68] Hansen D. Johnson, Mark F. Baumgartner, and Christopher T. Taggart. Estimating north atlantic right whale (eubalaena glacialis) location uncertainty following visual or acoustic detection to inform dynamic management. *Conservation Science and Practice*, 2(10):e267, 2020.

[69] R. Abileah. Use of high resolution space imagery to monitor the abundance, distribution, and migration patterns of marine mammal populations. In *MTS/IEEE Oceans 2001. An Ocean Odyssey. Conference Proceedings (IEEE Cat. No.01CH37295)*, volume 3, pages 1381–1387 vol.3, 2001.

[70] Caroline Höschle, Hannah C. Cubaynes, Penny J. Clarke, Grant Humphries, and Alex Borowicz. The potential of satellite imagery for surveying whales. *Sensors*, 21(3), 2021.

[71] Hannah Charlotte Cubaynes. *Whales from space: Assessing the feasibility of using satellite imagery to monitor whales*. PhD thesis, University of Cambridge, 2020.

[72] Hannah C. Cubaynes, Peter T. Fretwell, Connor Bamford, Laura Gerrish, and Jennifer A. Jackson. Whales from space: Four mysticete species described using new vhr satellite imagery. *Marine Mammal Science*, 35(2):466–491, 2019.

[73] Peter T. Fretwell, Iain J. Staniland, and Jaume Forcada. Whales from space: Counting southern right whales by satellite. *PLOS ONE*, 9(2):1–9, 02 2014.

[74] Emilio Guirado, Siham Tabik, Marga L Rivas, Domingo Alcaraz-Segura, and Francisco Herrera. Whale counting in satellite and aerial images with deep learning. *Scientific reports*, 9(1):1–12, 2019.

[75] Alex Borowicz, Hieu Le, Grant Humphries, Georg Nehls, Caroline Höschle, Vladislav Kosarev, and Heather J. Lynch. Aerial-trained deep learning networks for surveying cetaceans from satellite imagery. *PLOS ONE*, 14(10):1–15, 10 2019.

[76] Nabin Sharma, Paul Scully-Power, and Michael Blumenstein. Shark detection from aerial imagery using region-based cnn, a study. In Tanja Mitrovic, Bing Xue, and Xiaodong Li, editors, *AI 2018: Advances in Artificial Intelligence*, pages 224–236, Cham, 2018. Springer International Publishing.

[77] Tanya Y Berger-Wolf, Daniel I Rubenstein, Charles V Stewart, Jason A Holmberg, Jason Parham, Sreejith Menon, Jonathan Crall, Jon Van Oast, Emre Kiciman, and Lucas Joppa. Wildbook: Crowdsourcing, computer vision, and data science for conservation. *arXiv preprint arXiv:1710.08880*, 2017.

[78] Hansen Johnson, Daniel Morrison, and Christopher Taggart. Whalemap: a tool to collate and display whale survey results in near real-time. *Journal of Open Source Software*, 6(62):3094, 2021.

[79] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[80] Yan Liu, Qirui Ren, Jiahui Geng, and Meng Ding. Efficient Patch-Wise Semantic Segmentation for. pages 1–16, 2018.

[81] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[82] Barret Zoph, Ekin D. Cubuk, Golnaz Ghiasi, Tsung-Yi Lin, Jonathon Shlens, and Quoc V. Le. Learning data augmentation strategies for object detection, 2019.

[83] Rui Ma, Pin Tao, and Huiyun Tang. Optimizing data augmentation for semantic segmentation on small-scale dataset. In *Proceedings of the 2nd international conference on control and computer vision*, pages 77–81, 2019.

[84] Alexander Buslaev, Vladimir I Iglovikov, Eugene Khvedchenya, Alex Parinov, Mikhail Druzhinin, and Alexandr A Kalinin. Albumentations: fast and flexible image augmentations. *Information*, 11(2):125, 2020.

[85] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

[86] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors, 2012.

[87] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[88] Yichuan Tang and Chris Eliasmith. Deep networks for robust visual recognition. pages 1055–1062, 08 2010.

[89] Boyi Li, Felix Wu, Ser-Nam Lim, Serge Belongie, and Kilian Q. Weinberger. On feature normalization and data augmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12383–12392, June 2021.

[90] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[91] Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.

[92] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 818–833, Cham, 2014. Springer International Publishing.

[93] Chandramouli Shama Sastry and Sageev Oore. Detecting out-of-distribution examples with Gram matrices. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 8491–8501. PMLR, 13–18 Jul 2020.

[94] André Araujo, Wade Norris, and Jack Sim. Computing receptive fields of convolutional neural networks. *Distill*, 2019. https://distill.pub/2019/computing-receptive-fields.

[95] Marcelo Bertalmio, Andrea L Bertozzi, and Guillermo Sapiro. Navier-stokes, fluid dynamics, and image and video inpainting. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, pages I–I. IEEE, 2001.

[96] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[97] Frank Klawonn, Frank Höppner, and Balasubramaniam Jayaram. What are clusters in high dimensions and are they difficult to find? In Francesco Masulli, Alfredo Petrosino, and Stefano Rovetta, editors, *Clustering High–Dimensional Data*, pages 14–33, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[98] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[99] Andre G. C. Pacheco, Chandramouli S. Sastry, Thomas Trappenberg, Sageev Oore, and Renato A. Krohling. On out-of-distribution detection algorithms with deep neural skin cancer classifiers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2020.

[100] Bingyi Kang, Zhuang Liu, Xin Wang, Fisher Yu, Jiashi Feng, and Trevor Darrell. Few-shot object detection via feature reweighting. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[101] Bo Sun, Banghuai Li, Shengcai Cai, Ye Yuan, and Chi Zhang. FSCE: few-shot object detection via contrastive proposal encoding. *CoRR*, abs/2103.05950, 2021.

[102] Mengde Xu, Zheng Zhang, Han Hu, Jianfeng Wang, Lijuan Wang, Fangyun Wei, Xiang Bai, and Zicheng Liu. End-to-end semi-supervised object detection with soft teacher. *CoRR*, abs/2106.09018, 2021.

[103] Abdelrahman Abdelhamed, Marcus A. Brubaker, and Michael S. Brown. Noise flow: Noise modeling with conditional normalizing flows. *CoRR*, abs/1908.08453, 2019.

[104] Tianfu Wu and Xi Song. Towards interpretable object detection by unfolding latent structures. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6033–6043, 2019.

[105] Kira Vinogradova, Alexandr Dibrov, and Gene Myers. Towards interpretable semantic segmentation via gradient-weighted class activation mapping. *CoRR*, abs/2002.11434, 2020.

[106] Lukas Biewald. Experiment tracking with weights and biases, 2020. Software available from wandb.com.

Appendix A

Object to Background Ratio Calculation

We used generated masks from the two labeling schemes of elliptic and bounding box labeling to calculate the object to background ratio. In this project, we used the more accurate labeling scheme, elliptic labeling, for semantic segmentation whenever possible. Hence, to accurately calculate the ratio of object to the background, we used the same masks. To calculate the ratio, we added up the binary index of the pixels where they are not 0 (background) to get the summed value of pixels for the objects. Then we averaged this value over the spatial resolution of images (7360×4912) over 1544 images to get the mean ratio. The exact value for the ratio is 0.0016152578314526246, and algorithm 2 shows the approach we used for this calculation.

Algorithm 2: Object to Background Ratio Calculation

Input: List of $Masks$

Output: Object to Background $Ratio$

1: $ObjectPixelsSum = 0$
2: **for** $Mask \in Masks$ **do**
3: $\quad ObjectPixelsIndices = 1$ **Where** $Mask \neq 0$
4: $\quad ObjectPixelsSum = ObjectPixelsSum + ObjectPixelsIndices$
5: **end for**
6: Object to Background $Ratio = \frac{ObjectPixelsSum}{1544 \times 7360 \times 4912}$
7: **return** Object to Background $Ratio$

Appendix B

Data augmentation policies

The following is the complete list of data augmentation policies that we discussed in Chapter 5. These data augmentations are in Albumentation Notation. There is a web app[1] for illustrating the effect of augmentation in Albumentation.

B.1 Object Detection

B.1.1 Positive

- HorizontalFlip
- VerticalFlip
- ToGray
- RandomBrightnessContrast
- RandomRotate90
- ToSepia
- CLAHE
- FancyPCA
- JpegCompression
- Equalize
- VerticalFlip
- Rotate
- RandomBrightnessContrast
- Transpose
- RandomRotate90
- RandomRain
- RandomGamma
- RandomFog
- RGBShift
- MultiplicativeNoise
- MotionBlur
- JpegCompression

B.1.2 Negative

- HorizontalFlip

[1] https://albumentations-demo.herokuapp.com/

- ISONoise
- GridDistortion
- GaussNoise
- Equalize
- ElasticTransform
- Downscale
- CoarseDropout
- Cutout
- ChannelShuffle
- GaussianBlur
- GlassBlur
- Blur
- ShiftScaleRotate
- ToGray
- ToSepia
- RandomSunFlare
- CLAHE
- GridDistortion

B.2 Semantic Segmentation

- RandomHorizontalFlip
- RandomVerticalFlip
- ColorJitter
- RandomGrayscale
- Rotate
- GridDistortion
- JpegCompression
- Cutout
- MotionBlur
- RGBShift
- FancyPCA
- CLAHE
- Equalize

Appendix C

Models Hyper-parameters

In this appendix, we list the hyper-parameters we utilized for our model training.

C.1 Faster R-CNN

- Backbone: ResNet50 with FPN
- Anchor Generator: 3 Aspect ratios: 0.5, 1.0, 2.0 with 5 Scales: 32, 64, 128, 256, 512 $\rightarrow$ Total combinations: 15
- Pre-training: `COCO 2017`
- Fine-tuning Epochs: 30
- Training image resolution: $1200px \times 800px$
- Optimizer: SGD
  - Learning Rate: $5e-3$
  - Momentum: 0.9
  - Weight Decay: $5e-4$
- Learning Rate Scheduler:
  - Step: 3
  - γ : 0.1
- Batch Size: 8

C.2 DeepLabv3

- Backbone: ResNet101
- Pre-training: `COCO 2017`
- Fine-tuning Epochs: 40
- Training image resolution: $600px \times 400px$
- Optimizer:Adam
  - Initial Learning Rate: $1e-4$
  - betas: $(0.9, 0.999)$
  - Weight Decay: 0
- Learning Rate Scheduler:
  - Step: 5
  - γ : 0.1
- Batch Size: 4

Appendix D

Segmentation Mask to Bounding Box Conversion

Segmentation masks have a class value for each pixel in the input image, and the bounding box needs four corners of objects. We applied the **connected-component labeling** algorithm to segmentation masks to extract bounding boxes from them. This algorithm is derived from graph theory to extract sub-graphs that any two vertices are connected in a graph. Applying it to binary images gives a label for each region. A segmentation mask for C classes can be converted into C binary images. The indices where the input mask corresponds to a specific class label (L) comprise a binary image useful for a connected-component labeling algorithm to extract regions. When needed to convert segmentation masks to bounding boxes, we applied the connected-component labeling on the segmentation masks in a class-wise fashion to extract regions. The minimum and maximum values of x and y coordinates of these numbered regions are the bounding box corner coordinate. Hence the tuple of (X1 (x_{min}), X2 (x_{max}), Y1 (y_{min}), Y2 (y_{max}), L) is the extracted bounding box for each object. This procedure is depicted in figure D.1.

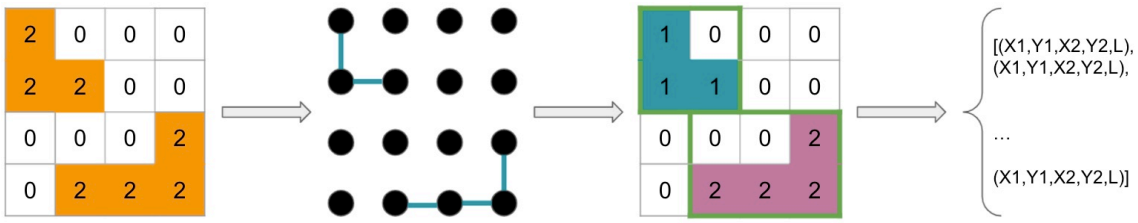


Figure D.1: Pixel-wise predictions to bounding box conversion (Bounding Box Extraction). An example input mask consists of regions with the same label that goes into the connected-component labeling algorithm to form two distinct labeled regions that provide the proper format for the output bounding boxes.

Appendix E

Metrics

Here, we discuss the metrics we used for evaluating models. Two common metrics have been widely used in competitions and academic research for object detection: PASCAL VOC and COCO. Both of these metrics are based on the concept of Intersection over Union.

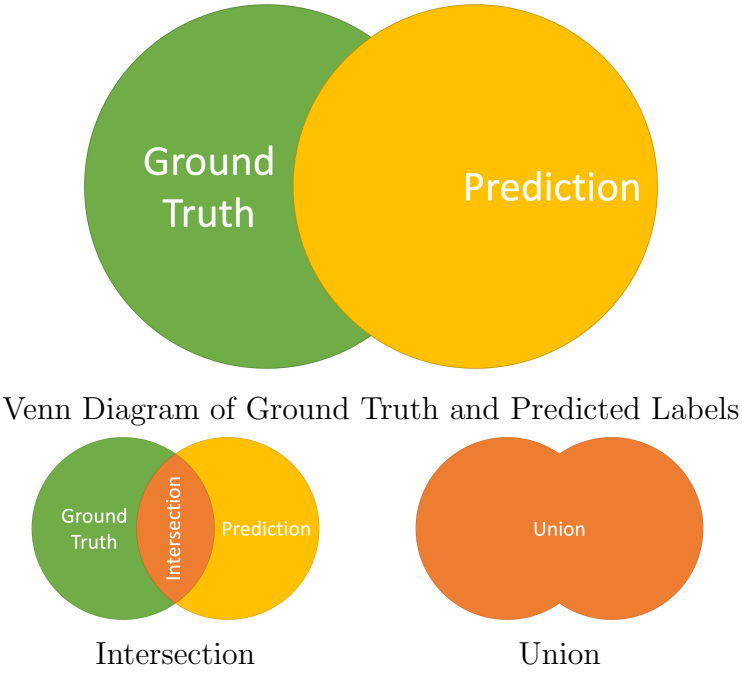


Figure E.1: Venn diagram of IoU

E.1 Intersection over Union

Intersection over Union (**IoU**), also known as the **Jaccard Index**, is a metric to analyze how well an object detection or a segmentation model performs. IoU is defined as the intersection of two sets (ground truth and model predictions) over their union. The Venn diagrams of these notions are depicted in figure E.1. If data is skewed in favor of the majority class, pixel-wise accuracy is not informative since

gathering the distribution information into a scalar value washes away the detail about the distribution. In these cases, IoU is a better metric because it computes the similarities between ground truth and prediction sets while it normalizes the scale of it based on the sizes of the objects.

E.2 COCO and PASCAL VOC Metrics

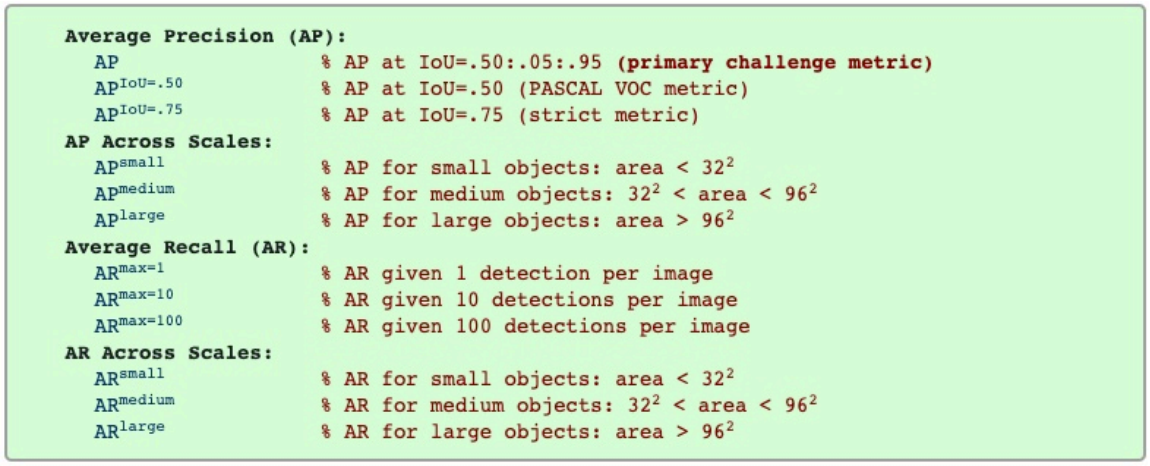


Figure E.2: COCO metrics. Taken from *COCO Detection Evaluation*[1]

In the case of having a confidence score threshold in the model, such as classifier score in Faster R-CNN, **true positive**, **false positive**, and **false negative** can be defined as below:

If a prediction has a higher confidence score than the threshold and the class matches the ground truth class with an IoU above the threshold, the prediction is a true positive. If a prediction has a higher confidence score than the threshold, but its class does not match the ground truth, or the IoU threshold is lower than the threshold, the prediction is a false positive. Finally, if the confidence score is lower than the threshold for an object, that prediction is a false negative. With these three, **precision** and **recall** can be computed. Since the precision and recall values are different for each confidence threshold, the interpolated area under the precision-recall curve for every confidence threshold is a standard metric, namely **average precision (AP)**. For multi-class object detection tasks, averaging AP over per class yields **mean**

[1] https://cocodataset.org/#detection-eval

average precision (mAP). Another numerical metric is **average recall (AR)**, which is twice the area under recall-IoU for IoU$>$ 0.5. PASCAL VOC metric is mAP when the IoU threshold is set to 0.5. COCO metrics and their descriptions are shown in Figure E.2.

E.3 Mean IoU (mIoU)

Mean IoU (mIoU) is a metric based on IoU to evaluate models for semantic segmentation task. Mean IoU is the average of IoUs computed for ground truth masks and predictions over all the classes.

Appendix F

Programming Details

Here is the list of technologies/libraries we used for this project.

F.1 Deep Learning Framework

- PyTorch: For model development alongside with Torchvision

F.2 Computer Vision Library

- OpenCV
- PIL

F.3 General Purpose Data Wrangling and Scientific Computing

- NumPy
- SymPy
- scikit-learn
- pandas
- Albumentations

F.4 Experimental Tracking

- Weights and Biases [106]

F.5 Image Annotation

- CVAT