A Comparison of Different Deep Learning Architectures on Coral Reef Detection

Mahdi Kashani, Brian Hopkinson, Ph.D, Prof. Suchendra Bhandarkar

Department of Computer Science, University of Georgia, email: mahdi@cs.uga.edu
Department of Computer Science, University of Georgia, email: sushi@uga.edu
Department of Marine Science, University of Georgia, email: bmhopkin@uga.edu

Keywords: Deep Learning, Machine Learning, Supervised Learning, Marine Science

1. Abstract

In this project I have conducted: 1- exploratory data analysis on our new coral-reef dataset and quality of bounding box annotation, 2- preprocessed the dataset 3- transfer learning (multistep based algorithm), 4- data augmentation, 5- examined four models, YOLOv2 and v3, Faster-RCNN and RetinaNet. First two are implemented in Darknet and the later ones are from Detectron2. I have trained each model with and without data augmentation. Finally, each technique is evaluated based on mAP, True Positive, True Negative and mIoU. YOLOv3 with mAP 0.35 and YOLOv2 with 0.28 rank the first the second. Unfortunately, we did not succeed to extract TP, TN and mIoU for Detectron2. In addition, we did not observe any significant increase by using data augmentation techniques. The report provides all the codes and the data analysis pipeline to reproduce the result. I hope that this dataset and reproducible comparison using all of the above platforms facilitates other researchers in multidisciplinary areas and environmental sciences, ecology, and marine data science.

2. Introduction and Literature Review

Distribution of coral reefs across the oceans over time is a good indicator of the impact of global warming and water pollution levels. Coral reefs do not tolerate temperature changes. A quarter of the carbon dioxide emission in the atmosphere is absorbed by the ocean. In addition, there has been a significant increase in ocean pollution(Carrington 2020; “Website” n.d.). This has resulted in the loss of 20% of the coral reef areas since the 1950s. (“IUCN (International Union for Conservation of Nature),” n.d.) (IUCN) red list of Threatened Species (“IUCN 2017” n.d.) Corals can be defined as the gaseous substances that provide the vital life-sustaining force are

one of many small components that belong to the long list of whys these organisms are so essential. Think about what makes up the earth at a glance. The oceans are prodigious in its makeup. Responsible for driving life as a whole, the coral reef systems cannot be ignored. Producing and filtering and harboring the most basic and immensely game-changing organisms, coral reefs as we know them, mark the course of life and earth that creates a picture of what we see as our ideal and functional planet.

Object detection refers to the methods to distinguish between objects recorded on the image. This technology has numerous applications, such as in automated cars, military usages, and identification. The capacity of problem-solving of artificial neural networks helped researchers to achieve accurate and fast systems that can detect items in each part of the picture correctly and provide a comprehensive knowledge of the image for the computer. In such a system, we classify the objects into prepared classes that have specific features for themselves. With processing the whole image, the machine will find these features and report the result.

With the recent advancement in Robotics and the deep learning technologies and growing interests in leveraging these technologies in marine science, there has been an urge to collect vast amounts of data as well as developing customized models for coral reef identification aiming to facilitate the study of marine life.

Since the early days researchers started to collect, annotate and build classifiers using image enhancement techniques and feature extractors (Beijbom et al. 2012), (Pizarro et al. 2008), (Stokes, Dale Stokes, and Deane 2009; Shihavuddin et al. 2013). (Viola and Jones, n.d.) used sliding windows as a simple convolutional operation to look over all of the boxes location and potentially do the classification task. Conventional object detection systems use a window to surf around the image and find objects (Papageorgiou and Poggio 2000) which is called the sliding window. Although it was not a new idea to implement the sliding window with convolution operation (Matan et al. 1992), this method carries out the computational cost. Even at that time of (Viola and Jones, n.d.). However, by time passing forward and thanks to many advances in technology and high-performance computations and parallel processing, cheaper memories that can exceed over thousands of IOPs it became possible to add more layers of convolutions and feature extraction.

Recently Convolutional Neural Networks have shown an outstanding result, where there is no

need to extract features neither the preprocessing steps are minimal (Krizhevsky, Sutskever, and Hinton 2017)(Russakovsky et al. 2015) However, still in the case of coral classification is challenging for many CNN methods. There is a significant variance between images, light variation, and the limitation in capturing all aspects and global forms of reefs, which makes it even challenging for human experts to agree on the identification. Two techniques have been practiced to overcome these challenges: transfer-learning (Tan, et al. 2018) and data augmentation (Shorten and Khoshgoftaar 2019).

2.3 What are the goals of the project?

While it is still an ongoing question for researchers in the domain of marine sciences to select appropriate models for their deep learning task, It is an ongoing question for practitioners of deep learning to choose the appropriate platform to satisfy a high-quality, high performance codebase for object detection research.

The goal of this project is to compare different networks performance on the provided coral reef data. I am curious given this data, what are the most suitable architectures, and given the limited data, what is the impact of data augmentation on these models. I developed a series of codes to reproduce and extend the work for future researcher. Finally, I share suggestions for the future work on this topic. Through this project, I learned how to setup the cloud environment to experiment these models and familiarized myself with various aspects of Deep Learning.

2.5. Why were these object detectors chosen?

Towards these goals, I have considered two major types of influential CNNs models; two Region-based methods, RetinaNet (Lin, Goyal, et al. 2017) and Faster R-CNN (Ren et al. 2015). I have used the Detectron2 implemented to explore these architectures. I also have used YOLO2 and YOLO3. YOLO is a popular model which is used widely in various application of computer vision.

Choosing RetinaNet: To initially guess which architecture works the best, we chose the top five images (based on number of objects) in Figure 5, in order of their object numbers in our dataset to put them into different pre-trained models based on the COCO dataset to see which pre-trained model is already working better. I found that among all of the architectures, *retinanet_R_50_FPN_1x* is proposing a good amount of locations. Table 10 summarizes these

tests, and links to the result. However, for later analysis we only used the *RetinaNet_R_50_FPN_1x*.

Name	lrschedule	NumWorker	Batch Size	Iteration	Learning Rate	Evaluation Rate	Link to the result!	TensorBoard
R50-C4	1x	4	64	300	0.001	200	comment	1th
R50-DC5	1x	4	64	300	0.001	200	comment:	2th
R50-FPN	1x	4	64	300	0.001	200	comment	3th
R50-C4	3x	4	64	1000	0.001	300	comment	4th
R50-DC5	3x	8	64	1000	0001	400	comment	5th
R50-FPN	3x	4	64	300	0.001	200	comment6	6th
R101-C4	3x	4	64	300	0.001	200	comment7	7th

Table 1: Testing with RetinaNet to see which architectures returns more objects.

At this step, I have not trained anything, and these state of art networks have never seen our dataset. However, in this experience, we hypothesized that the RetinaNet is going to give us the best performance in comparison to the other architectures that are already trained for fours on the COCO dataset and are not able to see any object for example, in Figure 1 the out result for the red object is Broccoli (one of the COCO dataset objects). Table 1, lists all type of RetinaNet I have tested.

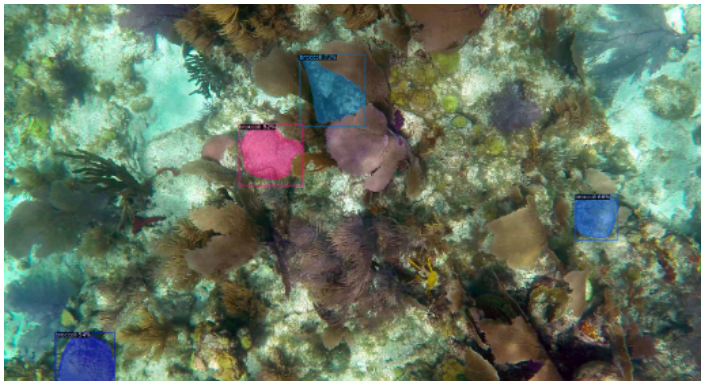


Figure 1: Using pretrained RetinaNet to see how many items are detected. It was interesting to see that pre-trained mask_rcnn_R_50_FPN can do the segmentation as well as proposing the bounding box and detecting them as broccoli (because it's only trained on coco). However, it is also sad to see that there is nothing around those two stones above that are detected as broccoli. Even the "Past" on the right side of the image that is detected as a brucceli seems to be chewed by fishes.

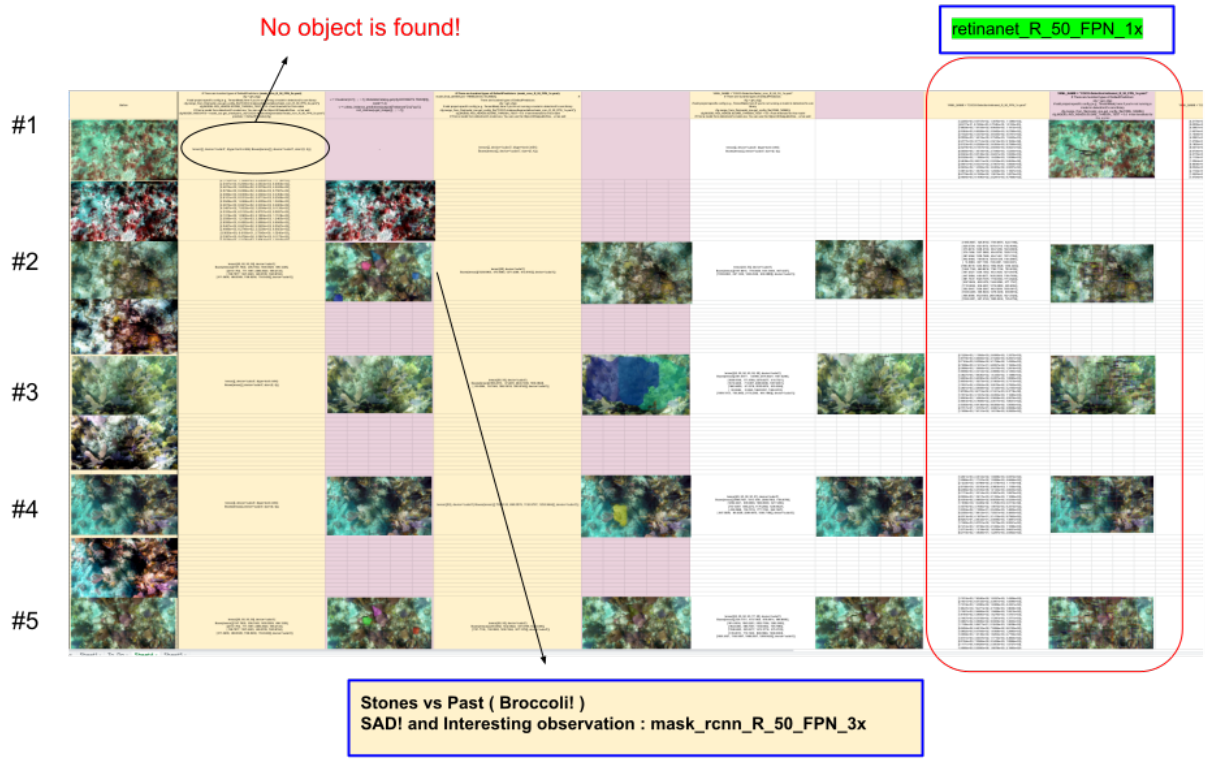


Figure 2: Initial pre-trained model selection before transfer learning and preprocessing

2.6 Provide a brief description of the object detectors used with references to the literature.

Region-based methods

These methods are based on the principle of scanning images through sliding windows. These networks consist of a layer or a subpart to produce Region of Interest (ROIs) for object detection. Various models within this set of methods have a different strategy to optimize the analysis of ROIs either faster or more accurately, for example, through a Selective Search (Uijlings et al. 2013) or Boundary Box Regressors (Uijlings et al. 2013; Dickerson, n.d.) or ROI Pooling (Girshick 2015).

Region-based convolutional network (R-CNN) (Girshick et al. 2016) proposed to overcome the drawbacks of previous methods and for accurate object detection and segmentation. It uses the minimized number of windows provided by selective search then it extracts features with CNN. This system can classify each window using an SVM classifier.

The R-CNN was time-consuming, so (Girshick 2015) proposed a new method called Fast R-

CNN to overcome the problem. In this method, we turn the image to features and find the proposed window from it. So, we turn the image to features once, on the contrary, in R-CNN windows turned to features each time, and the computation speed was so high. These windows will feed fully connected layers and softmax layers at last as an activator function to map the previous results to percentages for probability and determination.

Non-Region-based methods or single shot detectors

These methods unlike the region-based models do not use regions to localize objects with the picture. In YOLO (Liu et al. 2016) a single CNN predicts the boundary boxes and the class. they utilize default boxes, which can surely be designed according to the case. Also, they come with the flexibility of multi-scaling features that improve the performance when objects are in different sizes. Still, these changes compromised the accuracy with speed and showed to have lower performance when objects are smaller or tend to be placed closer to each other. These methods are designed for object detection in real-time. These methods have a shorter GPU time but might have lower accuracy (Huang et al. 2017). Figure 1 from (Huang et al. 2017) maps various models into two axes of accuracy and computation time.

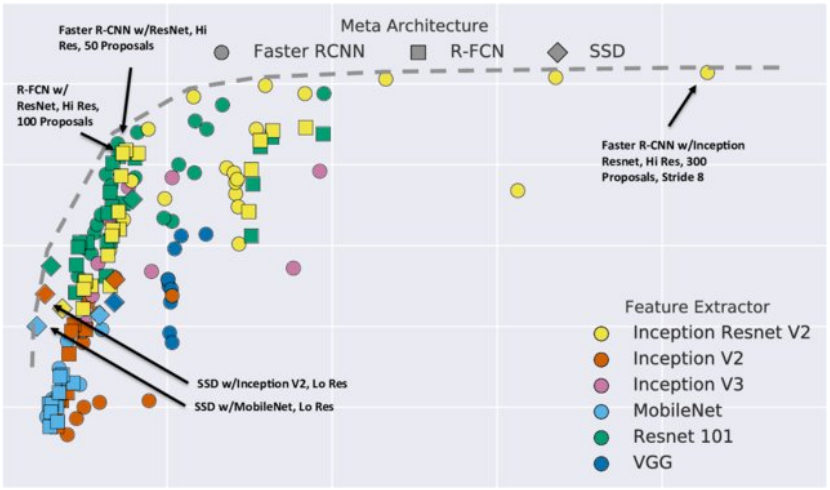


Figure 3: From Huang et. Al 2017: Speed versus accuracy trade-off for modern Convolutional Object Detectors.

Therefore, I decided to two models from each group and investigate how well each fit to our case. Given the very few data points and our imbalanced data I also aimed to evaluate the data augmentation technique on each model to see how this technique can improve the performance

and generalization.

I have used the Detectron2 to explore the region-based models. Detectron project (Girshick et al. 2018) includes implementation for various object detection algorithms including Mask R-CNN (Kaiming et al. 2017) -- Marr Prize at ICCV 2017, RetinaNet (Lin, Goyal, et al. 2017) -- Best Student Paper Award at ICCV 2017, Faster R-CNN (Ren et al. 2015), Fast R-CNN (Girshick 2015), R-FCN (Dai et al. 2016) while providing various backbone network architectures including ResNeXt{50,101,152} (Xie et al. 2017) , ResNet{50,101,152} (He et al. 2016) Deep Residual Learning for Image Recognition, Feature Pyramid Networks (Lin, Dollár, et al. 2017) for object detection (with ResNet/ResNeXt), and VGG16 (Simonyan and Zisserman 2014) which is very Deep Convolutional Networks for Large-Scale Image Recognition. Detectron was written in python and powered by the Caffe2 deep learning framework. Coffe2 is built on the original Caffe but with expression, speed, and modularity in mind.

(2019) Detectron2 (Wu et al. 2019); however, is the new ground-up rewrite of Detectron (Girshick et al. 2018) in PyTorch and it is still an on developing next-generation software system that has implemented the state-of-the-art object detection algorithms and it roots from, but it's not limited to maskrcnn-benchmark (Massa and Girshick 2018) which was already written in Pytorch as a successful rapid solution for instance segmentation and object detection algorithms and it is deprecated since Detectron2 includes all of the maskrcnn architectures which extends Faster RCNN for predicting an object mask in parallel with recognizing the bounding box for those objects. It also includes DensePose (Alp Güler, Neverova, and Kokkinos 2018) that can dense correspondences from two dimensional images to surface-based representation and better scene understanding that might be useful in underwater habitat. It also includes panoptic feature pyramid networks (Kirillov et al. 2019) proposed by Alexander Kirillov as a robust and more accurate baseline for both instance and semantic segmentation.

RetinaNet: RetinaNet is a single-stage object detector developed by (Lin, Goyal, et al. 2017) This algorithm finds objects in different scales via a feature pyramid network. This method uses a deep learning core called Resnet after FPN. They proposed a new loss function called focal loss, which focuses on hard negative instances instead of easy negatives.

YOLO: As a novel strategy, You Only Look Once (YOLO) represented by (Redmon et al. 2016). Similar to RetinaNet it can evaluate an image one time and report class probability and

object boundaries. This algorithm divides input into grids. One of the most drawbacks of this system is its limit in the recognition of close objects. Linear regression helps YOLO to report the final result. The interior architecture is based on 24 convolutional neural network layers with a sum-squared error loss function.

---

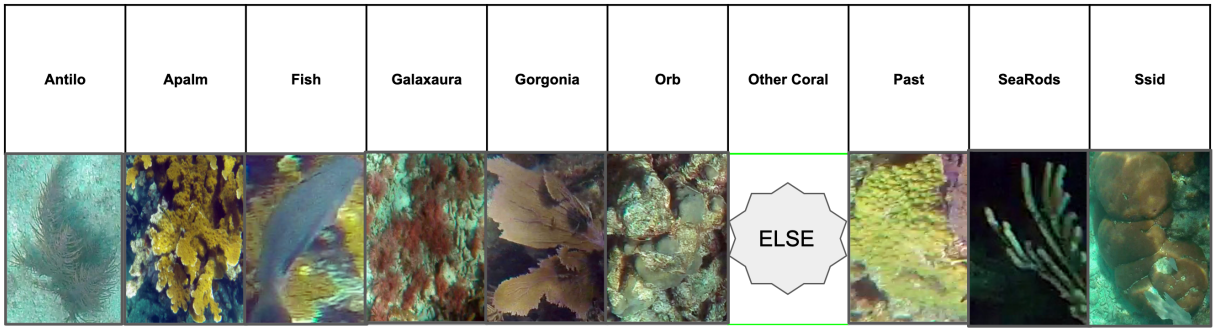
3. Materials and Methods (800 - 1000)

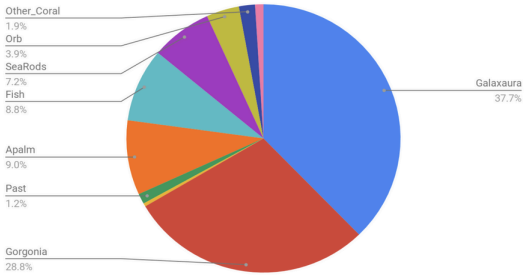
3.1-Data

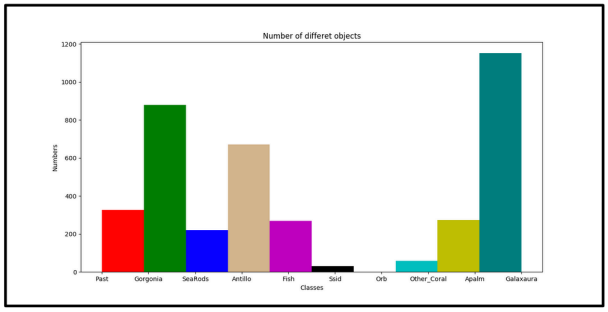
The data consists of 133 annotated images, from 10 different classes of reefs in a JPEG format, together with an annotation file in a comma separated file. Images are labeled by experts using (microsoft VoTT) in Hopkinson Lab. The annotation file details the coordinates of the boundary boxes of each object in each image along with the object's name. The annotation only includes the upper right and the lower left side of each box as it is shown in the Figure 3.

Each image includes approximately 10 different objects and the size of each unique class of object varies across images. Table 2 describes the size and the number of objects in the dataset. Figure 2 displays each of these 10 classes as well as the histogram of objects occurrence in the entire dataset.

It is important to explore the data beforehand and assess aspects like number and frequency of these objects across datasets. This analysis helps to diagnose and interpret each model's performance more effectively. Objects with a few numbers of occurrences are naturally harder to predict. Since there has not been enough samples to learn their pattern. Also, it is important to ensure the training set includes all available objects, otherwise, obviously, the trained model will not be able to identify unseen objects. The size of objects can also be interesting to assess and later on correlate with the performance of each model. In fact, we expect to see variance in result according to the object size. Since all object detection models work on the idea of boxes, whether through a brute-force search as in region-based model or predefined boxes in singe-shot detectors. The Table 4 lists these properties within each class.







	Percentage
Galaxaura	1151
Gorgonia	880
Antillo	12
Past	38
Apalm	274
Fish	270
SeaRods	221
Orb	118
Other_Coral	58
Ssid	30

Figure 4: The data consists of 133 annotated images. There are 10 different classes in the dataset. Classes are not equally occurring in images.

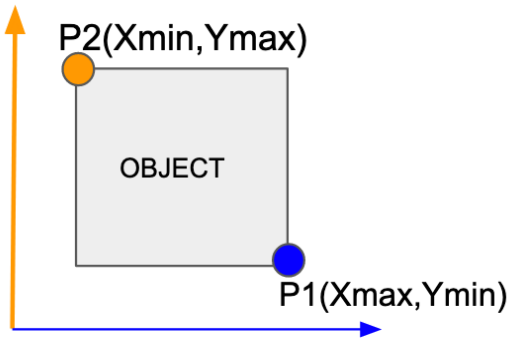


Figure 5: Objects are in different sizes; therefore, we have calculated the average area size for each objects along with the variance in the area size.

There have been several challenges in data preprocessing

1) The annotation file is not consistent, meaning there are identical images with two different annotation. For example, file: 3D_L0443_112 which results in error like the following in Figure 4.

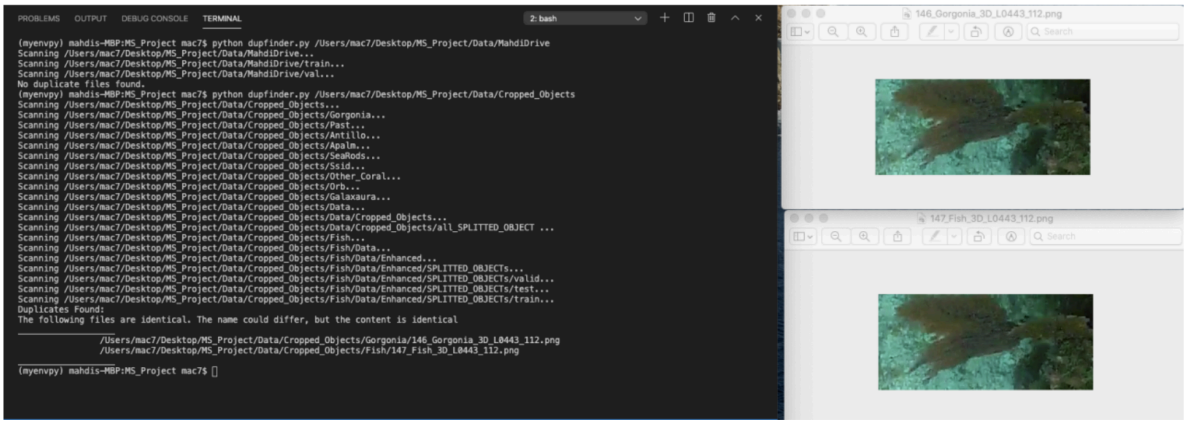


Figure 6: Identical images with different annotation files

2) Labeling is contaminated

The training data does not have a good quality and expect this to impact our performance, therefore, I tried to re-annotate these images using pixel-based annotation. Pictures 6 and 7 show examples.

3) The dataset is not fully annotated. As Figure 5 shows, there are files with only two objects while other files include 56 objects.

4) Changing the format of the annotation file. Most of the libraries and state-of-art-codebases are working with JSON format and they need that JSON format to be in either COCO format or VOC.

top 5 images	Number of Objects
A_3D_L0646_144.jpg	56
3D_L0622_176.jpg	55
3R010215_829.jpg	54
B_3D_L0647_42.jpg	53
3D_L0622_139.jpg	53

bottom 5 images	Number of Objects
3D_L0443_21.jpg	8
3D_L0443_36.jpg	5
3D_L0441_41.jpg	4
3D_L0457_97.jpg	2
3D_L0453_119.jpg	2

Figure 7: The annotation file for this image include a box as a fish

Figure 8: shows annotations do not cover the whole picture. They also have overlap.

Figure 9: Annotation can include multiple objects. The box include one Galaxuara however, it includes at least three of them. Such annotation will negatively impact the result of the model.

.

Class name	Number of occurrences in dataset (1)		Present in % of images (2)		Average number of class per image (3)		Average area size in pixel (4)		Variability in area size In pixel (5)	
	#	Rank	#	Rank	#	Rank	#	Rank	#	Rank
Past	326	4	40.6	5	2.5	4	24K	9	43K	8
Gorgonia	880	2	78.2	2	6.6	2	65K	4	72K	4
SeaRods	221	7	69.9	3	1.7	7	41K	8	38K	9
Antillo	671	3	84.2	1	5.1	3	59K	5	70K	5
Fish	279	5	54.9	4	2.0	6	12K	10	19K	10
Ssid	30	10	15.8	10	0.2	10	90K	3	113K	3
Orb	118	8	34.6	7	0.9	8	323K	1	514K	1
Other_coral	58	9	18.1	9	0.4	9	58K	7	67K	6
Apalm	274	6	21.8	8	2.1	5	218K	2	49K	2
Galaxaura	1151	1	36.1	6	8.7	1	58K	6	64K	7

Table 2:A brief overview on the datasets. The description of each column is as following: 1) number of occurrences in the data indicates how rare an object is. There are not many Ssid in the entire dataset. 2) This column indicates whether objects across images

3.2. Describe the training procedure(s) used:

I used 10% of the data, equal to 13 images for testing and the rest for the training. Potentially, I could have used some of the comparable data from other datasets. However due to the limitation in time and access to domain experts we continue the experiment with only 13 images as test. It is worth mentioning, all the testing data is the same for all models to correctly compare their performance. Table 3 shows class are distributed across the training and the test set.

category	#instances	category	#instances	category	#instances
Past	304	Gorgonia	841	SeaRods	200
Antillo	613	Fish	220	Ssid	29
Orb	76	Other_Coral	57	Apalm	274
Galaxaura	990				
total	3604				

Table 3: Distribution of instances in our training set and test set. We should make sure to include from each class both training and evaluation.

Preprocessing:

Through our experimentation with RetinaNet, the first model I started with, I realized the images do not look sharp, even hard for human to identify the objects. Therefore, I started experimenting images by increasing the quality. My preprocessing includes histogram equalization. This process normalizes the darkness and brightness in the image. This is a common preprocessing filter to improve the image qualities. Finally, I used color balancing. This filter adjusts the intensity of the colors. These filtering, improve the image quality and naturally increase the object identification even for human. As a simple test, without training any model, we observed that a pretrained RetinaNet model on the COCO dataset, detect more objects on these enhanced images, compared to raw images. Figure 10 displays an example.

Data Augmentation:

YOLO and Detectron both have parameters to set to augment data internally. These augmentations are listed in the Table 4. I noticed YOLO do not have flipping. So, I do the flipping beforehand. This means, first we flip all 120 images for training (90% of the data) then, we use these original enhanced images, and flipped images to train the deep learning model. As I mentioned, each codebase has an internal process. This means, the code base randomly decides to do any of the operation to augment the data and grantee to augment by 50% of the data. This is probabilistic, because the internal pipelines augment images randomly. So for non-augmented models we have 120 images. For augmented models we have 120+120+(<randomly produced images between 120 and 240).

	Operation
YOLO	• Saturation = 1.5 • Exposure = 1.5 • Hue = 0.1 • Jitter = 0.3 • Min_crop=224 • Max_crop=448 • Rotation(max) = 90
Detectron2	• RandomFlip() • ResizeShortestEdge() • RandomBrightness=(0.8,1.2) • RandomCrop(crop_type='relative_range', crop_size=(0.8, 0.8)) • RandomSaturation(0.8,1.2) • RandomRotation([0,90])

Table 4: Yolo and Detectron2 both offer functionality to augment data. However, there is no one to one map. We have used the default value for each of them. The data augmentation occurs within the batch. Pictures are randomly assigned to these operation. In additional, I have made and used 120 flipped images.

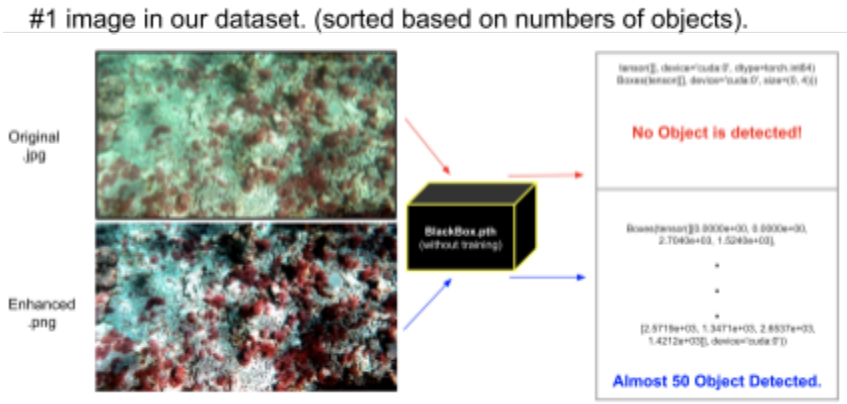


Figure 10: The importance of preprocessing and how it might change the results. We realized enhancing the image quality help the neural network to detect more objects.

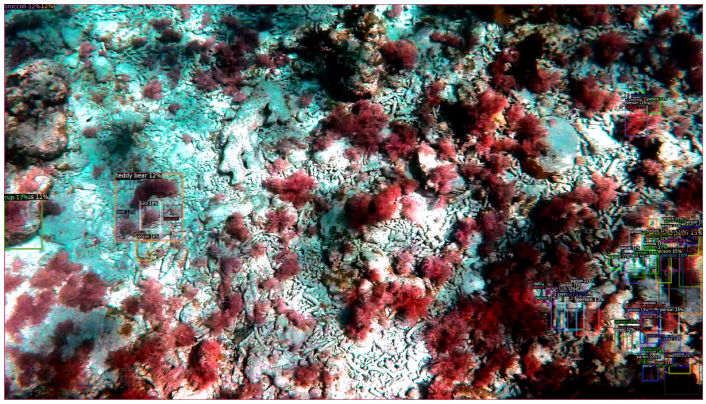


Figure 11: It's worth saying that Galaxaura is annotated for almost 33 percent of our dataset; but at this point we have decided to evaluate our initial hypothesis by finally training our state of art architectures to solve this problem

Transfer Learning and training pretrained models:

For each model, we have trained a model with and without data augmentation to investigate the impact on the result. The training of each Yolo model took from 15 to 18 hours using GPU using Amazon Cloud GPU P2.Xlarge (0.9$/hour) and Simple Storage Service S3. I have moved all the polished notebooks and data into Colab as the GPU service is for free and easier to share. So the current testing set up is there. The training for each model based on Detectron2 took from 10 to 12 hours using GPU on Google Colab. I have used the all default parameters as they were recommended for each codebase. This include the learning rate, activation function, batch size, subdivision.

YOLO object detection is officially only trained in *ImageNet* architecture. Therefore, we cannot tweak with the number of neurons or number of convolutional layers/nodes, the total number of parameters would change from the official one, so the pre-trained weights would not load for different numbers of parameters. Therefore, no hyperparameters are possible to be configured for training a specific YOLO model as the formula for the last layer filters is specific ((num_of_classes+5)*num_of_masks) which is constant for each version of YOLO and we cannot change the node count of previous layers as then we will not be able to use pre-trained weights.

Validation procedure(s) used

We used 10% of our data as a test set to evaluate our models. These images are exactly the same for all the models, to ensure all models are tested equally regardless of difficulty of the task. We used Average Precision (AP), True Positive and True Negative to assess these models.

We also tried to extract mean Interaction over Union, however Detectron2 does not return this value. In our study, I have calculated the Average Precision per category for every model that I have trained. Then I have averaged over all of our ten groups in our study to calculate the primary mAP.

Later we added number of false-positive, false negatives, precision, recall, accuracy using the confusion matrix, and the following formula in the Figure 12

True_Positive	False_Positive	Predicted											
False_Negative	True_Nagative	Antilo	Apalm	Fish	Galaxaura	Gorgonia	Orb	Other Coral	Past	SeaRods	Ssid	False_Negative	Recal
Actual	Antilo	A	Ab	Ac	Ad	Ae	Af	Ag	Ah	Ai	Aj	Ab + Ac + Ad + Ae + Af + Ag + Ah + Ai + Aj	A / A + (FN)
	Apalm	BA	B	Ba	Bd	Be	Bf	Bg	Bh	Bi	Bj	BA + Ba + Bd + Be + Bf + Bg + Bh + Bi + Bj	B / B + (FN)
	Fish	CA	CB	C	Cd	Ce	Cf	Cg	Ch	Ci	Cj	CA + CB + Cd + Ce + Cf + Cg + Ch + Ci + Cj	C / C + (FN)
	Galaxaura	DA	DB	DC	D	De	Df	Dg	Dh	Di	Dj	DA + DB + DC + De + Df + Dg + Dh + Di + Dj	D / D + (FN)
	Gorgonia	EA	EB	EC	ED	E	Ef	Eg	Eh	Ei	Ej	EA + EB + EC + ED +Ef + Eg + Eh + Ei + Ej	E / E + (FN)
	Orb	FA	FB	FC	FD	FE	F	Fe	Fh	Fi	Fj	FA+ FB + FC + FD + FE + Fe + Fh + Fi + Fj	F / F + (FN)
	Other Coral	GA	GB	GC	GD	GE	GF	G	Gh	Gi	Gj	GA + GB + GC + GD + GE + GF +Gh + Gi + Gj	G / G + (FN)
	Past	HA	HB	HC	HD	HE	HF	HG	HH	Hi	Hj	HA + HB + HC + HD + HE + HF + HG + Hi + Hj	H / H + (FN)
	SeaRods	IA	IB	IC	ID	IE	IF	IG	IH	I	Ij	IA + IB + IC + ID + IE + IF + IG + IH + Ij	I / I + (FN)
	Ssid	JA	JB	JC	JD	JE	JF	JG	JH	JI	J	JA + JB + JC + JD + JE + JF + JG + JH + JI	J / J + (FN)

	False_Positive	(BA) + (CA) + (DA) + (EA) + (FA) + (GA) + (HA) + (IA) + (JA) .	(Ab) + (CB) + (DB) + (EB) + (FB) + (GB) + (HB) + (IB) + (JB) .	(Ac) + (Ba) + (DC) + (EC) + (FC) + (GC) + (HC) + (IC) + (JC) .	(Ad) + (Bd) + (Cd) + (ED) + (FD) + (GD) + (HD) + (ID) + (JD) .	(Ae) + (Be) + (Ce) + (DE) + (FE) + (GE) + (HE) + (IE) + (JE) .	(Af) + (Bf) + (Cf) + (Df) + (Ef) + (GF) + (HF) + (IF) + (JF) .	(Ag) + (Bg) + (Cg) + (Dg) + (Eg) + (Fe) + (HG) + (IG) + (JG) .	(Ah) + (Bh) + (Ch) + (Dh) + (Eh) + (Fh) + (Gh) + (IH) + (JH) .	(Ai) + (Bi) + (Ci) + (Di) + (Ei) + (Fi) + (Gi) + (Hi) + (JI) .	(Aj) + (Bj) + (Cj) + (Dj) + (Ej) + (Fj) + (Gj) + (Hj) + (Ij) .
	Precision	A / A + (FP)	B / B + (FP)	C / C + (FP)	D / D + (FP)	E / E + (FP)	F / F + (FP)	G / G + (FP)	H / H + (FP)	I / I + (FP)	J / J + (FP)
	Accuracy	Sum(Green) / Whole_Chart									

Figure 12: How we can calculate the False_Positive & False_Negative in our multi class confusion matrix

4. Experimental Results (1000 - 1500)

We have summarized the value of the models in the following table. Table 5 summarizes the performance (AP, TP and FP, mIoU[1]) of each model on each class as well as the running time. Table 6 shows the performance as well as how much Data Augmentation had improved the result. As you can see data augmentation in general has not increased the performance Later you can see the illustrated results on some of the images.

I have further used a few samples to test the segmentation task using Mask-RCNN. Because, the results are not thorough, I only have put a few result in the Appendix section.

Detectron2 fails to return object coordinates to calculate TP, FP mIoU.

There was no explicit function of feature to output the mIoU, TP or FP in Detectron2 package. So, to calculate it I tried first to edit their evaluation script i.e. pycoco tool's *cocoeval.py* but their format even the shape of array in very different and got no docs about it, so I tried a lot to understand there format and got nothing after debugging and writing codes. Then I tried to just take out the detection boxes from detectron2, first, it wasn't allowing me to compile. Install that code, anyhow I compiled it then got the boxes array with so many parameters not just xmin, xmax and min, ymax, so we didn't know which column represents which parameter. even the shape of array was too large. Then, I tried this link: https://github.com/facebookresearch/detectron2/issues/586 and tried to calculate those with the file specified in the link. It wasn't accepting the box array got from detectron2 previously as the data representation needed and the data we got aren't matching.

[1] Detectron2 does not return mIOU and unfortunately, I did not find any way to extract these values. YOLO uses it own evaluation scripts to show more metrics than official ones. Detectron2 uses the PyCOCO API in official from COCO datasets and it only evaluate on mAP, AP and recall nothing else is outputted. It is only available for the segmentation task.

Table 5: Performance of the five CNN deep learning architecture. We have listed Average Precision (AP), True Positive (TP) and False Positive (FP) for each model.

Name	YOLOv2						YOLOv3						Faster_RCNN						RetinaNet					
	Augmented			Non-Augmented			Augmented			Non-Augmented			Augmented			Non-Augmented			Augmented			Non-Augmented		
	AP	TP	FP	AP	TP	FP	AP	TP	FP	AP	TP	FP	AP	TP	FP	AP	TP	FP	AP	TP	FP	AP	TP	FP
Antillo	0.42	25	10	0.39	24	16	0.30	18	11	0.42	26	11	0.26	-	-	0.15	-	-	0.26	-	-	0.13		-
Apalm	0	0	1	0	0	1	0	0	0	0	0	1	0.0	-	-	0.0	-	-	0.0	-	-	0.0		-
Fish	0	0	3	0.02	1	3	0.02	1	4	0.05	2	4	0.08	-	-	0.07	-	-	0.04	-	-	0.10		-
Galaxaura	0.22	53	108	0.24	62	123	0.29	56	67	0.26	62	102	0.08	-	-	0.11	-	-	0.07	-	-	0.12		-
Gorgonia	0.21	10	13	0.29	13	13	0.29	7	4	0.40	13	5	0.18	-	-	0.18	-	-	0.18	-	-	0.20		-
Orb	0.18	7	2	0.14	6	7	0.19	6	2	0.19	5	5	0.17	-	-	0.12	-	-	0.19	-	-	0.11		-
Other_Coral	0	1	0	1	1	0	1	1	0	1	1	0	0.0	-	-	0.0	-	-	0.0	-	-	0.12		-
Past	0.35	8	5	0.52	12	6	0.47	9	4	0.61	14	5	0.19	-	-	0.19	-	-	0.26	-	-	0.26	-	-
SeaRods	0.38	6	0	0.16	3	3	0.23	1	1	0.28	5	2	0.11	-	-	0.20	-	-	0.10	-	-	0.15	-	-
Ssid	0	0	1	1	0	0	0	0	0	0	0	1	0.0	-	-	0.0	-	-	0.0	-	-	0.0	-	-
mAP	0.1756			0.3761			0.2791			0.3214			0.1167			0.1126			0.1224			0.1339		
mIoU	0.2840			0.2863			0.3582			0.3580			NA[2]			NA			NA			NA		
time	15-18 h			15-18 h			15-18 h			15-18 h			10-12 h			10-12 h			10-12 h			10-12 h		

[2] Detectron2 does not return mIOU and unfortunately, I did not find any way to extract these values. YOLO uses it own evaluation scripts to show more metrics than official ones. Detectron2 uses the PyCOCO API in official from COCO datasets and it only evaluate on mAP, AP and recall nothing else is outputted. It is only available for the segmentation task.

Run time

Run time for training is listed in the table above. In general, for YOLO it is around 15-18 hours whereas for Detectron2 is 10-12 hours. This is the time when the total loss stops changing. The GPU kernels die sometimes because they are running in parallel. Therefore, I had to see which one was died and stopped at which iteration, so it was hard to extract more detailed training time for all these models. Testing is very fast and does not make much time. The running time on test test varies across images, but in general we noticed the YOLO models run faster compare to the to other models. In genral regardless of the model the testing time is fast. Because the weights are calculated. Also, we are testing only on 10% of the data. The augmentation data does not add much into the running time. Because the data is still small. Detectron2 is faster than Yolo due to the efficient use of CUDA cores and data structure.

The effect of Augmentation

Since the data is small, I decided to explore the effect of data augmentation on my models. Augmentation did not improve the performance significantly among the models and it led to a lower performance. Given our objects have not been seen in the training dataset, and the complex large size of these networks, most likely the augmentation leads to further overfitting. This behavior has been reported before (Shorten and Khoshgoftaar 2019) where the Augmentation in domains with very limited data, has led to further overfitting. In addition, the augmentation strategy for YOLO and Detectron looks slightly different as it is shown in the Table 6.

	Mean IoU Non-augmented	Mean Iou Augmented	mAP Non-augmented	mAP Augmented	Δ mAP
YOLO v2	0.2863	0.2840	0.376	0.1756	-53%
YOLO v3	0.3580	0.3582	0.321	0.2791	-13%
Faster R-CNN	NA	NA	0.113	0.117	+4%
RetineNet	NA	NA	0.134	0.122	- 9%

Table 6: Summary of how much data augmentation improve or worsen the mean average precision for each Model across the objects. In overall, regardless of the class type, data augmentation worsens the performance of single shot detectors and improve the performance. Detectron2 does not return mIoU in detection task.

Illustration of the result from five different models on three different images.

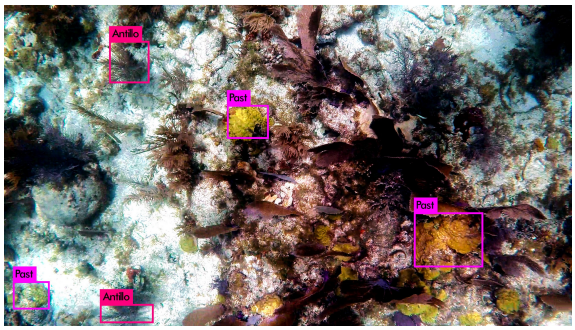
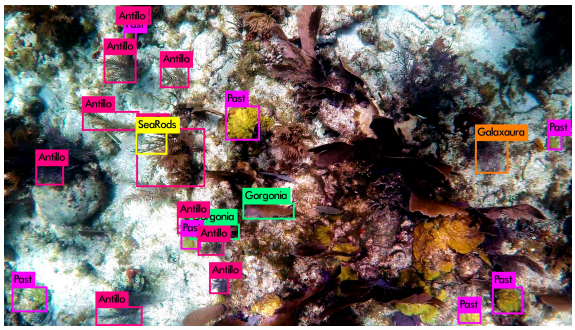
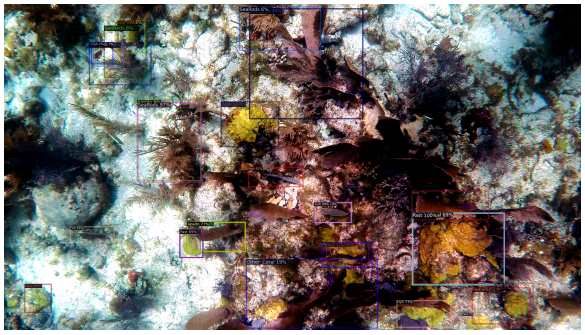
Yolo 2	Yolo 3
	
Faster-RCNN	**RetinaNet**
	

Illustration of the result from five different models on three different images.

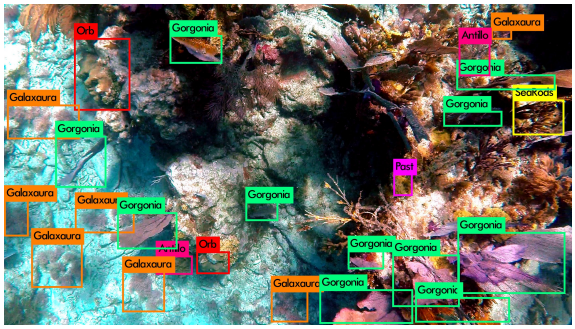
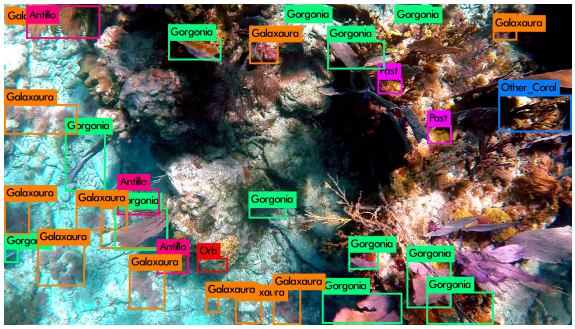
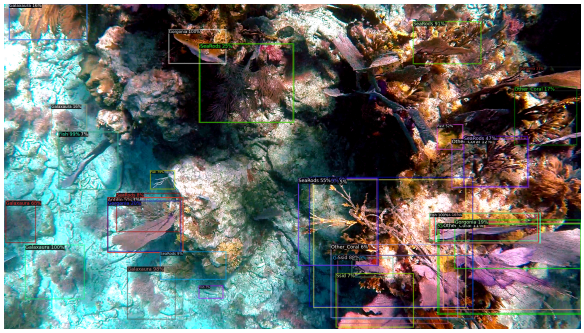
Yolo 2	Yolo 3
	
Faster R-CNN	**RetinaNet**
	

Illustration of the result from five different models on three different images.

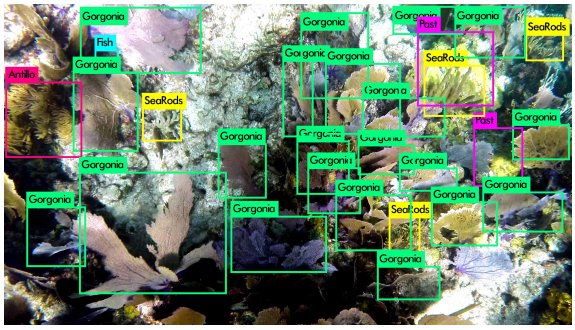
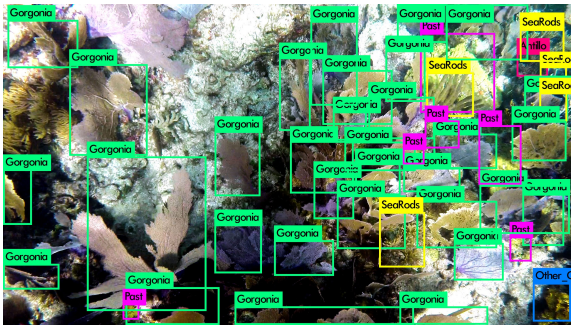
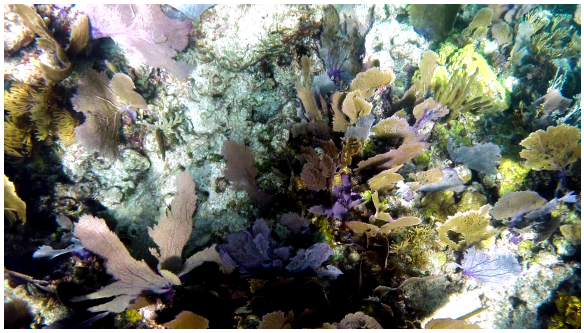
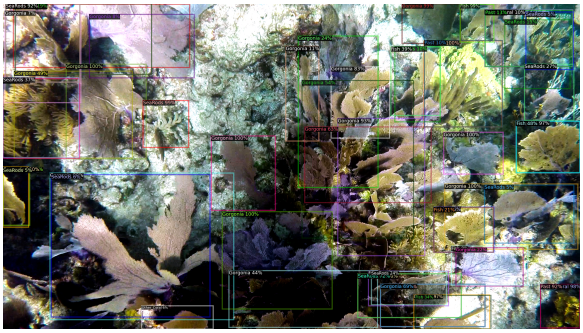
Yolo 2	Yolo 3
	
Faster R-CNN	**RetinaNet**
	

Figure 7: Table 5Delta Average Precision (AP) shows how much data augmentation has increased or decreased the average precision

Name	YOLOv2			YOLOv3			Faster_RCNN			RetinaNet		
	AP augmentation	AP non-augmentation	Δ AP	AP augmentation	AP non-augmentation	Δ AP	AP augmentation	AP non-augmentation	Δ AP	AP augmentation	AP non-augmentation	Δ AP
Antillo	0.42	0.39	+ 8%	0.30	0.42	- 29 %	0.26	0.15	+73%	0.26	0.13	+100%
Apalm	0	0	0%	0	0	0%	0.0	0.00	0%	0.00	0.00	0%
Fish	0	0.02	- 100%	0.02	0.05	- 60 %	0.08	0.07	+14%	0.04	0.10	-60%
Galaxaura	0.22	0.24	- 8%	0.29	0.26	+12 %	0.08	0.11	-27%	0.07	0.12	-42%
Gorgonia	0.21	0.29	- 28%	0.29	0.40	- 28 %	0.18	0.18	0%	0.18	0.20	-10%
Orb	0.18	0.14	+ 29%	0.19	0.19	0 %	0.17	0.12	+42%	0.19	0.11	+73%
Other_Coral	0	1	- 100%	1	1	0 %	0.0	0.0	0	0.00	0.12	-100%
Past	0.35	0.52	-33%	0.47	0.61	- 23 %	0.19	0.19	0%	0.26	0.26	0%
SeaRods	0.38	0.16	+137%	0.23	0.28	- 18%	0.11	0.20	-45%	0.10	0.15	-33%
Ssid	0	1	-100%	0	0	0 %	0.0	0.0	0%	0.00	0.00	0%
mAP	0.18	0.38	-53%	0.28	0.32	- 13 %	0.12	0.11	+9%	0.12	0.13	-8%

Objects can be challenging for some and easy for some others.

Regardless of the model Apalm stays undetectable and data augmentation does not help either. These classes are the largest objects by area after orbs. Ssids are also very difficult to predict and only YOLO2 is able to detect them. These groups of objects are also the least frequent classes in the data. There exist only 30 of them in the entire dataset. So the impact of imbalanced data is more sever on this class of objects.

Past has the highest average AP across the models. Antilo has the second best average AP, which could be because there are 5 of this class almost per image. Therefore, enough sample size to learn for models. The same reason can explain the relatively high average AP of Gorgona. These can support the major issue with this dataset is the its size.

YOLOv3 and YOLOv2 have the highest mAP 0.32 and 0.37. YOLO models are unable to detect fish. YOLO works on the principle of looking at the full image once but the networks we trained. However, the other region-based networks are working on fundamentals similar to the convolution's window sliding and pooling. The YOLO emphasizes on region of interest after looking at the picture and is probably capable to cover the full body of Past class but the window protocol may not be able to cover the full body in a window in all images, that's why maybe recognize them as another or no class. Still the dataset is small, and most observations stay at the speculation level.

5. Conclusions

Here we have analysed a coral reef data. I extensively explored the annotation data as it was not in the best quality. We realized there are issues in the annotation process that should be improved in the future. We set up an AWS pipeline, including the storage as well as cloud computation and SageMaker to perform this analysis using GPU capabilities. To share the result we had to move the notebooks and data into Google Colab.

The number of images and the quality of annotation are major issues. We had to clean the data, correct the annotation manually to make the model work. We also started doing further annotation which could be used for segmentation. Since the data was very small for such a task, we used Data Augmentation tactic and we observed it had improved the model for some of the models. In terms of running run we can see the YOLO models takes longer, but in return the perform better.

I compared multiple object detection algorithms. We observed YOLO do well particularly on larger objects like Pest. However, it seems the data augmentation leads to a lower performance, potentially due to over fitting. Data Augmentation had also negatively impacted the region-based algorithms and the result was worse for some classes than the other. This result suggest we still need manual data annotation and we cannot compromise that with data augmentation.

Last but not the least, we have provided pipelines using popular codebases, Detectron2 and YOLO for future researchers to reproduce our result as well as extend this investigation more efficiently. In our postprocessing we tried to calculate the performance of each models. Produce boundary boxes on the test images to visualize the result.

Future work:

New annotation for the segmentation task

As we mentioned there annotation data suffer from bad quality. In future, aligned with increasing the quality of the annotation, we can create annotation data specific for segmentation. As we have briefly showed in the Appendix.

Using other datasets:

There are a few datasets when it comes to coral reefs. ELIAT is a coral dataset containing 1123 patches from larger images acquired from coral reefs in the Red Sea (Gómez-Ríos et al. 2019). RSMAS is a smaller data consisting of 766 image patches from larger images acquired by different cameras in different places by the Rosenstiel School of Marine and Atmospheric Sciences of the University of Miami. Table 1 is a modified table from (Gómez-Ríos et al. 2019) summarizes classes as well as number of images per class.

Source Code

The project initially was located https://github.com/m-kashani/MS_Project. The repository includes all the details, including issues, errors. However, later, we have moved the final notebooks and libraries, for the sake of clarity here https://github.com/m-kashani/coral7200. We kept the older repository as it serves as a log for all day to day tasks during the project

. Appendix

Guideline to compile the report:

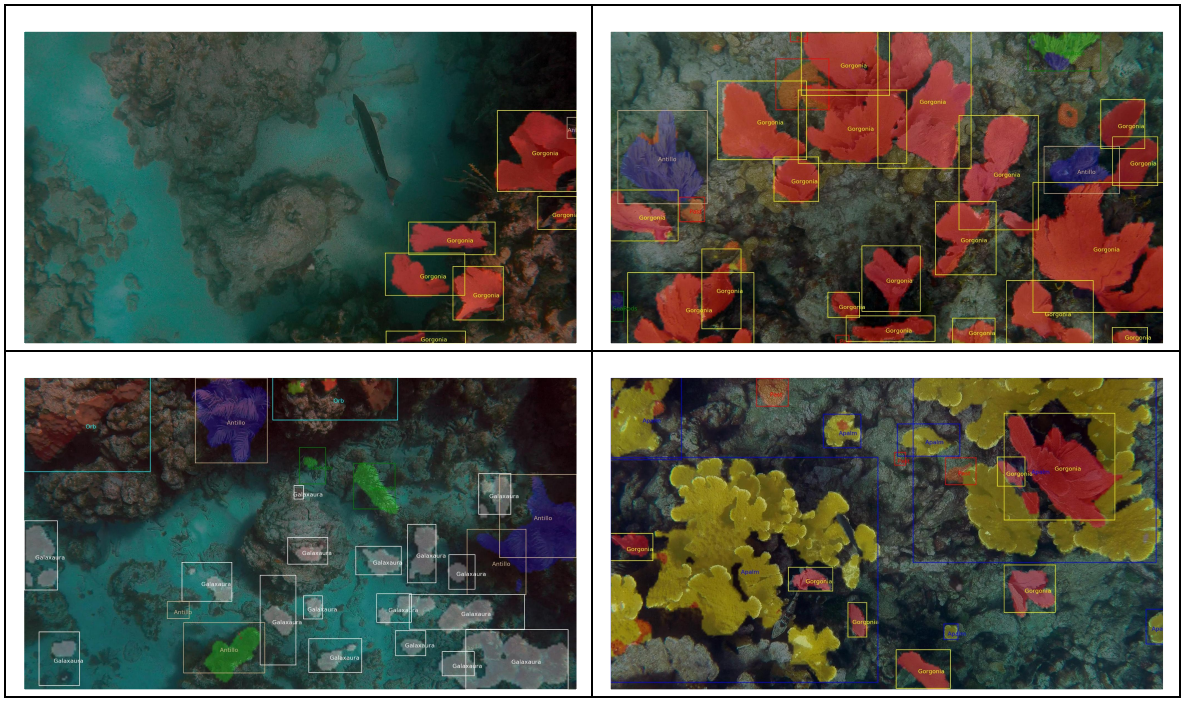
Project Technical Report Guidelines

1. **Project Summary** (250-300 words)
    1. Mention the goals of the project
    2. Mention and briefly describe the object detectors used.
    3. Mention and briefly describe the results obtained
    4. Outline the major conclusions of the project
2. **Introduction and Literature Review** (500-700 words)
    1. What is the project about?
    2. Why is important?
    3. What are the goals of the project?
    4. What were the techniques (i.e., object detectors) used?
    5. Why were these object detectors chosen?
    6. Provide a brief description of the object detectors used with references to the literature.

    For points 1–3 above, you may summarize the relevant material from Andrew King's papers and thesis.
3. **Materials and Methods** (800-1000 words)
    1. Describe the dataset used for training the object detectors.
    2. Describe the training procedure(s) used.
    3. Describe the validation procedure(s) used.
    4. Describe the performance metrics used to evaluate the results
        - False positive rate (FPR)
        - False negative rate (FNR)
        - Mean Intersection-over-Union (MIOU) for localization error
4. **Experimental Results** (1000-1500 words)
    1. Tabulate the above performance metrics (FPR, FNR and MIOU) for each of the object detectors. You may also use graphs in addition to the tables.
    2. Tabulate run times for training and testing for each of the object detectors.
    3. Show images with the object detection results for each of the object detectors for each coral reef class object.
5. **Conclusions** (500-700 words)
    1. Compare the object detectors in terms of the performance metrics (FPR, FNR and MIOU)
    2. Compare the object detectors in terms run times for training and testing.
    3. Mention any tradeoffs between the performance metrics (FPR, FNR and MIOU) and the run times for training and testing.
6. **Appendix**
    1. Provide information on the source code and the GitHub site from where it can be downloaded.

    Source Code: Provide the documented source code on GitHub with downloading and compiling instructions.

Sample annotation was done in pursuit of segmentation

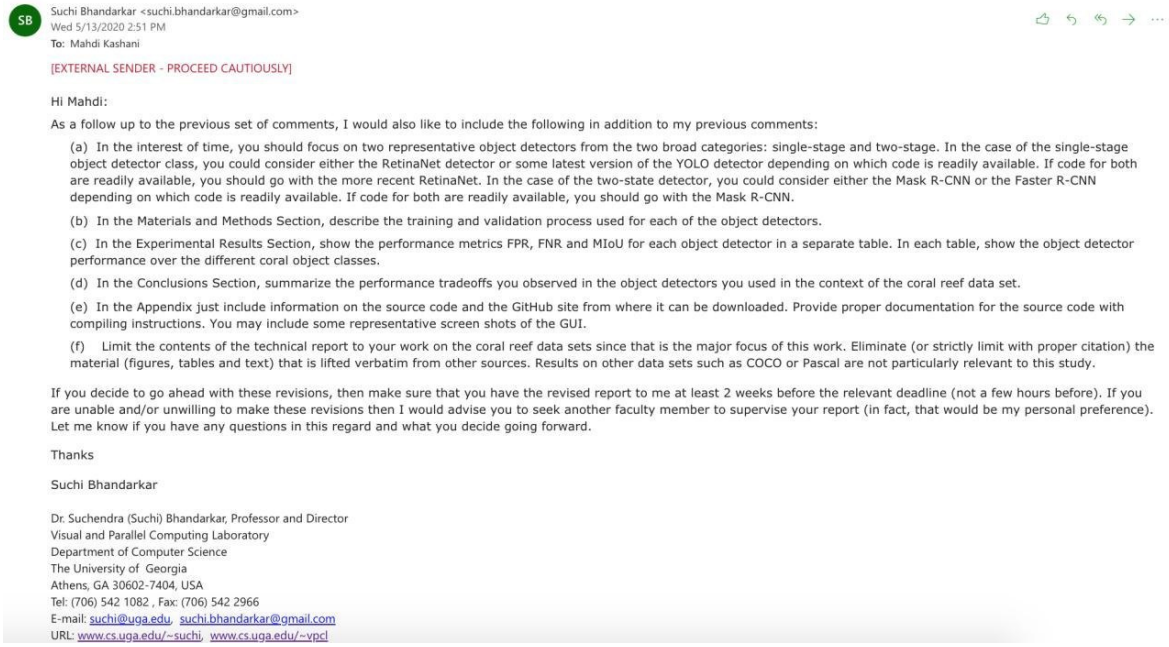


REFERENCES:

1. Alp Güler, Rıza, Natalia Neverova, and Iasonas Kokkinos. 2018. "Densepose: Dense Human Pose Estimation in the Wild." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 7297–7306.
2. Beijbom, O., P. J. Edmunds, D. I. Kline, B. G. Mitchell, and D. Kriegman. 2012. "Automated Annotation of Coral Reef Survey Images." *2012 IEEE Conference on Computer Vision and Pattern Recognition*. https://doi.org/10.1109/cvpr.2012.6247798.
3. Carrington, Damian. 2020. "Ocean Temperatures Hit Record High as Rate of Heating Accelerates." The Guardian. The Guardian. January 13, 2020. http://www.theguardian.com/environment/2020/jan/13/ocean-temperatures-hit-record-high-as-rate-of-heating-accelerates.
4. Dai, Jifeng, Yi Li, Kaiming He, and Jian Sun. 2016. "R-FCN: Object Detection via Region-Based Fully Convolutional Networks." In *Advances in Neural Information Processing Systems 29*, edited by D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, 379–87. Curran Associates, Inc.
5. Girshick, Ross. 2015. "Fast R-Cnn." In *Proceedings of the IEEE International Conference on Computer Vision*, 1440–48.
6. Girshick, Ross, Jeff Donahue, Trevor Darrell, and Jitendra Malik. 2016. "Region-Based Convolutional Networks for Accurate Object Detection and Segmentation." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 38 (1): 142–58.
7. Girshick, Ross, Ilija Radosavovic, Georgia Gkioxari, Piotr Dollár, and Kaiming He. 2018. "Detectron."
8. Gómez-Ríos, Anabel, Siham Tabik, Julián Luengo, A. S. M. Shihavuddin, Bartosz Krawczyk, and Francisco Herrera. 2019. *"Towards Highly Accurate Coral Texture Images Classification Using Deep Convolutional Neural Networks and Data Augmentation."* Expert Systems with Applications. https://doi.org/10.1016/j.eswa.2018.10.010.
9. He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. "Deep Residual Learning for Image Recognition." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 770–78.
10. Huang, Jonathan, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, et al. 2017. "Speed/accuracy Trade-Offs for Modern Convolutional Object Detectors." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 7310–11.
11. "IUCN 2017." n.d. List Table of Number of Threatened Species by Major Groupsof Organism. Accessed May 11, 2020. http://cmsdocs.s3.amazonaws.com/summarystats/2017-3_Summary_Stats_Page_Documents/2017_3_RL_Stats_Table_1.pdf.
12. "IUCN (International Union for Conservation of Nature)." n.d. https://doi.org/10.1163/1570-6664_iyb_SIM_org_2285.
13. Kaiming, He, Gkioxari Georgia, Dollar Piotr, and Girshick Ross. 2017. "Mask R-Cnn." In *International Conference on Computer Vision*. Vol. 1.
14. Kirillov, Alexander, Ross Girshick, Kaiming He, and Piotr Dollár. 2019. "Panoptic Feature Pyramid Networks." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 6399–6408.
15. Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. 2017. "ImageNet Classification with Deep Convolutional Neural Networks." *Communications of the ACM* 60 (6): 84–90.

16. Lin, Tsung-Yi, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. 2017. “Feature Pyramid Networks for Object Detection.” In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2117–25.
17. Lin, Tsung-Yi, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. 2017. “Focal Loss for Dense Object Detection.” In *Proceedings of the IEEE International Conference on Computer Vision*, 2980–88.
18. Lin, Tsung-Yi, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. 2014. “Microsoft COCO: Common Objects in Context.” In *Computer Vision – ECCV 2014*, 740–55. Springer International Publishing.
19. Liu, Wei, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. 2016. “SSD: Single Shot MultiBox Detector.” In *Computer Vision – ECCV 2016*, 21–37. Springer International Publishing.
20. Massa, Francisco, and Ross Girshick. 2018. “Maskrcnn-Benchmark: Fast, Modular Reference Implementation of Instance Segmentation and Object Detection Algorithms in PyTorch.”
21. Matan, Ofer, Christopher J. C. Burges, Yann Lecun, and J. S. Denker. 1992. “Multi-Digit Recognition Using a Space Displacement Neural Network.” In *Neural Information Processing Systems (NIPS 1991), San Mateo, CA*. Morgan Kaufmann.
22. Papageorgiou, Constantine, and Tomaso Poggio. 2000. “A Trainable System for Object Detection.” *International Journal of Computer Vision* 38 (1): 15–33.
23. Pizarro, Oscar, Paul Rigby, Matthew Johnson-Roberson, Stefan B. Williams, and Jamie Colquhoun. 2008. “Towards Image-Based Marine Habitat Classification.” *OCEANS 2008*. https://doi.org/10.1109/oceans.2008.5152075.
24. Redmon, Joseph, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. “You Only Look Once: Unified, Real-Time Object Detection.” In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 779–88.
25. Ren, Shaoqing, Kaiming He, Ross Girshick, and Jian Sun. 2015. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.” In *Advances in Neural Information Processing Systems 28*, edited by C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, 91–99. Curran Associates, Inc.
26. Russakovsky, Olga, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, et al. 2015. “ImageNet Large Scale Visual Recognition Challenge.” *International Journal of Computer Vision*. https://doi.org/10.1007/s11263-015-0816-y.
27. Shihavuddin, A. S. M., Nuno Gracias, Rafael Garcia, Arthur Gleason, and Brooke Gintert. 2013. “Image-Based Coral Reef Classification and Thematic Mapping.” *Remote Sensing*. https://doi.org/10.3390/rs5041809.
28. Shorten, Connor, and Taghi M. Khoshgoftaar. 2019. “A Survey on Image Data Augmentation for Deep Learning.” *Journal of Big Data* 6 (1): 60.
29. Simonyan, Karen, and Andrew Zisserman. 2014. “Very Deep Convolutional Networks for Large-Scale Image Recognition.” *arXiv [cs.CV]*. arXiv. http://arxiv.org/abs/1409.1556.
30. Stokes, M. Dale, M. Dale Stokes, and Grant B. Deane. 2009. “Automated Processing of Coral Reef Benthic Images.” *Limnology and Oceanography: Methods*. https://doi.org/10.4319/lom.2009.7.157.
31. Tan, et al. 2018. “*A Survey on Deep Transfer Learning.*” arxiv.org/pdf/1808.01974v1.pdf.

32. Viola, P., and M. Jones. n.d. “Rapid Object Detection Using a Boosted Cascade of Simple Features.” *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*. https://doi.org/10.1109/cvpr.2001.990517.
33. “Website.” n.d. Accessed May 11, 2020. https://arxiv.org/pdf/1808.01974v1.pdf.
34. Wu, Yuxin, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. 2019. “Detectron2.”



SB

Suchi Bhandarkar <suchi.bhandarkar@gmail.com>
Wed 5/13/2020 2:51 PM
To: Mahdi Kashani

[EXTERNAL SENDER - PROCEED CAUTIOUSLY]

Hi Mahdi:

As a follow up to the previous set of comments, I would also like to include the following in addition to my previous comments:

(a) In the interest of time, you should focus on two representative object detectors from the two broad categories: single-stage and two-stage. In the case of the single-stage object detector class, you could consider either the RetinaNet detector or some latest version of the YOLO detector depending on which code is readily available. If code for both are readily available, you should go with the more recent RetinaNet. In the case of the two-state detector, you could consider either the Mask R-CNN or the Faster R-CNN depending on which code is readily available. If code for both are readily available, you should go with the Mask R-CNN.

(b) In the Materials and Methods Section, describe the training and validation process used for each of the object detectors.

(c) In the Experimental Results Section, show the performance metrics FPR, FNR and MIoU for each object detector in a separate table. In each table, show the object detector performance over the different coral object classes.

(d) In the Conclusions Section, summarize the performance tradeoffs you observed in the object detectors you used in the context of the coral reef data set.

(e) In the Appendix just include information on the source code and the GitHub site from where it can be downloaded. Provide proper documentation for the source code with compiling instructions. You may include some representative screen shots of the GUI.

(f) Limit the contents of the technical report to your work on the coral reef data sets since that is the major focus of this work. Eliminate (or strictly limit with proper citation) the material (figures, tables and text) that is lifted verbatim from other sources. Results on other data sets such as COCO or Pascal are not particularly relevant to this study.

If you decide to go ahead with these revisions, then make sure that you have the revised report to me at least 2 weeks before the relevant deadline (not a few hours before). If you are unable and/or unwilling to make these revisions then I would advise you to seek another faculty member to supervise your report (in fact, that would be my personal preference). Let me know if you have any questions in this regard and what you decide going forward.

Thanks

Suchi Bhandarkar

Dr. Suchendra (Suchi) Bhandarkar, Professor and Director
Visual and Parallel Computing Laboratory
Department of Computer Science
The University of Georgia
Athens, GA 30602-7404, USA
Tel: (706) 542 1082 , Fax: (706) 542 2966
E-mail: suchi@uga.edu, suchi.bhandarkar@gmail.com
URL: www.cs.uga.edu/~suchi, www.cs.uga.edu/~vpcl