A Deep Learning Approach to Safeguard Coral Reefs: Detecting Crown-of-Thorn Starfish in Underwater Footage

Ken K. Hong, Oanh Doan
Georgia Institute of Technology
khong18@gatech.edu
odoan@gatech.edu

Abstract

The Great Barrier Reef, a UNESCO World Heritage site, faces severe threats from climate change, environmental pollution, and the overpopulation of Crown-of-Thorn Starfish (COTS). This project aims to improve real-time COTS detection in underwater footage using a deep learning-based object detection model, YOLOv11. Trained on a dataset of 23,501 underwater images with bounding box annotations, the model achieved a mean average precision (mAP50) of 0.729 and mAP50-95 of 0.331. With an inference speed of 7.0 ms per image (143 frames per second), the model shows satisfactory performance for real-time COTS detection, contributing to coral reef protection efforts.

1. Introduction/Background/Motivation

The Great Barrier Reef is the world's largest coral reef ecosystem and a UNESCO World Heritage site, hosting over 400 coral species. It is one of Earth's richest and most complex ecosystems. However, due to various factors, including climate change, environmental pollution, seawater warming, and attacks by Crown-of-Thorn Starfish (COTs), this coral reef ecosystem faces severe threats [5]. In particular, the overpopulation of COTS in Australia's Great Barrier Reef has caused significant damage to the coral and poses a serious threat to its delicate ecosystem.

Our primary goal is to develop an object detection model using deep learning model to assist in real-time detection of COTs from underwater footage. We will implement YOLOv11 by Ultralytics [4] as a baseline model and conduct hyper-parameter tuning to optimize model performance. In addition, we will integrate Convolutional Block Attention Model (CBAM) [6] to enhance feature extraction by focusing on important regions of the images. A successful implementation of this model would contribute to the efforts to detect and control COTS outbreaks in protection of the Australia's marine life.

Prior to our deep-learning models, various techniques, such as the 'Manta Tow' [1], have been deployed to detect and remove these starfish to protect marine life. Started in the 1960, manta tow technique is primarily based on human observation and subjective estimation, with no computational modeling or automated detection involved. In this method, a trained snorkel diver, while being towed by a boat on the water surface, visually assesses and records the presence and abundance of COTS along with other reef metrics. While generally effective, these methods face limitations in scalability, image resolution, reliability, and traceability.

Our project uses the Crown-of-Thorns Starfish (COTS) dataset, which was compiled through a collaboration between CSIRO's Data61, CSIRO Oceans & Atmosphere, Queensland University of Technology, and Google [2]. The dataset consists of Great Barrier Reef underwater footage with bounding box annotations indicating the location of COTS. The dataset includes images obtained from three distinct video IDs, each recorded in different underwater environments. Each image or each frame is uniquely identified by a combination of video ID and intra-video frame number. Some images may or may not contain COTS, resulting in images without any bounding box annotations. In total, the dataset comprises of 23501 RGB images with a resolution of 1280×720 pixels. Detailed information regarding the dataset is provided in Table 1.

Video ID	Total Frames	Frames with Bounding Boxes	% of Annotated Frames
0	6708	2143	31.95%
1	8232	2099	25.50%
2	8561	677	7.91%
Total	23501	4919	20.9%

Table 1: Data Distribution of Annotated Frames per Video

In addition, there are a few factors that differentiate this dataset with conventional object detection datasets [2]. First, since the objective is to detect COTS in underwater images, the COTS dataset contains the single COTS class. Second, the dataset naturally exhibits sequence-based annotations as multiple images capture the same COTS from slightly different angles or positions. Lastly, it includes overlapping COTS and partial visibility due to their cryptic behavior, where parts of the animal might be hidden.

2. Approach

2.1. Data Processing

In this project, we first performed exploratory data analysis to examine the dataset in detail, as described in the background section. Next, we addressed the issue of label format incompatibility. The original dataset provided bounding box annotations using the pixel coordinates of the upper-left corner (x_min, y_min) along with the box's width and height in pixels. We modified this setup to meet YOLO's required format, which uses the coordinates of the bounding box's center (x_center, y_center) and normalizes all coordinates and box measurements with respect to the original image's width and height. Hence, all normalized values are bounded between 0 and 1.

In addition to label conversion, we filtered the dataset to include only frames containing annotations, as unlabeled frames did not provide valuable information for training. Frames without bounding boxes often act as negative examples, which, when excessive, can lead to unbalanced learning, degraded model performance, and computational inefficiency. According to Table 1, applying this filter results in a dataset of 4919 images for both training and validation.

For the training and fine-tuning processes, $video0$ and $video1$ (4242 images) are used as the training dataset , and $video2$ (677 images) is reserved for validation . This dataset splitting approach provides a more reliable evaluation of the object detection model's performance and helps reduce the risk of overfitting to the training data. Additionally, instead of a random 80/20 split, a video-based split is used to prevent data leakage, as images from the same video sequence shows high similarity. Figure 1 visualizes one instance of the input data.

2.2. Model

We chose to use YOLOv11 [4] by Ultralytics, a state-of-the-art object detection model known for outperforming others in accuracy and efficiency. YOLOv11 is a particularly good choice for object detection tasks due to its real-time inference capabilities and superior performance in detecting small and intricate objects.

We began our experiments with the YOLOv11 nano model, which is optimized for lightweight, efficient training and inference. We trained the model for 10 epochs using default hyper-parameters and an input image size of 640. The model resulted in a mAP50 of 0.3, which was a promising signal of the model's potential. Building on these results, we decided to scale up to YOLOv11 small as our baseline model and experiment with larger models, including YOLOv11 medium, large, and even X-large. We hypothesize that larger models, equipped with more complex layers, are better at capturing low level details, which eventually would lead to improved model performance. This is particularly important for our task, where the starfish blend seamlessly into the coral reef and have subtle features that can be difficult to distinguish. Apart from the model size, we also tested for various hyper-parameters, including the use of cosine learning rate scheduling and the Adam optimizer with weight decay. A detailed discussion of hyper-parameter tuning experiments is provided in Section 3.1.

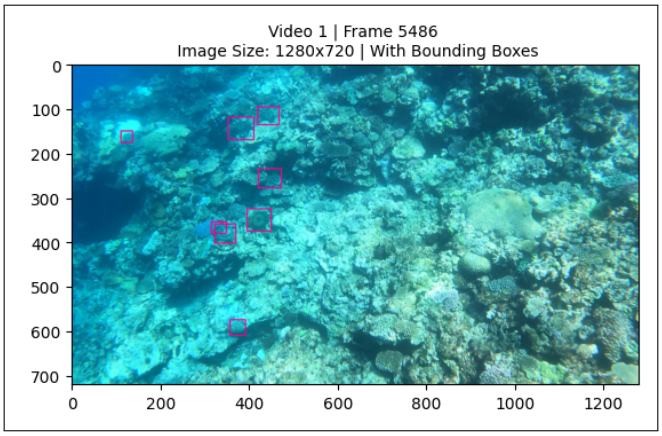

Figure 1: Input Data: An Annotated Frame with Bounding Boxes

Beyond tuning hyper-parameters to optimize the baseline model, we attempted two techniques to further enhance model performance. First, we added dummy classes to output layer as a form of regularization. We magnified the number of prediction classes, but no actual labels were created for these dummy classes. We expected this technique to reduce over-fitting and force the model to generalize better to unseen data. Second, we integrated the Convolutional Block Attention Module (CBAM) to refine the feature extraction process. Specifically, we inserted CBAM into the first two C3K2 layers in YOLO models, which are part of the model backbone and are responsible for feature extraction. With this update, CBAM is expected to process the output of the original C3K2 layers to suppress irrelevant information before passing it on to the next layer. We chose to insert CBAM to only the first two C3K2 layers because of two main reasons. First, they operate at early stages of the network, where broader and lower-level features are extracted. Second, the architecture of these two layers are

the same, which reduces friction in implementation. We opted not to insert CBAM into deeper layers in the network to minimize computational and implementation overhead, particularly when these layers tend to focus on higher-level, abstract features, where the need for attention refinement is less critical.

3. Experiments and Results

3.1. Experiment Setup

Computational resource is a key limiting factor in this project. The maximum GPU available on Google Colab (A100 with 40 GB memory) is not sufficient to train models with large image sizes, batch sizes, and model configurations. We acknowledge that the most systematic approach to tune hyper-parameters is to utilize the `model.tune()` [3] function provided by Ultralytics. However, due to resource constraint, we could not pursue this direction. Instead, we decided to override several hyper-parameters with selected values based on our own research. A detailed explanation of the overridden values is provided in table 2. We used these values as the new 'default' for subsequent tuning of hyper-parameters listed in Table 3, with the exception of Model 5b.

To begin with, we tested with a baseline YOLOv11 small model using an image size of 640 and a batch size of 8. Given the nature of our task, it is intuitive that higher image resolutions will enable the model to learn finer details that might otherwise be lost with low resolution images. Thus, we increased the image size to 1280 for all subsequent experiments.

Following the baseline model evaluation, we conducted fine-tuning of the models by varying batch sizes and introducing dummy classes. Batch size impacts gradient stability during training; smaller batch sizes may lead to noisier updates, while larger batch sizes provide more stable gradients but demand greater GPU memory. Additionally, dummy classes were introduced in some configurations to mitigate overconfidence in the model's predictions. Finally, we incorporated CBAM into the best-performing model configuration.

3.2. Experiment Results

Table 3 presents the experiment results. We use mAP50 and mAP50-95 as the primary metrics to evaluate model performance. mAP50 measures the number of good predictions, i.e. the number of predicted boxes that overlap reasonably well (50% IoU or more) with the true boxes. Compared to mAP50, mAP50-95 is a stricter accuracy measurement. It evaluates the model's ability to predict boxes at various levels of accuracy, from a rough match (50% IoU) to a near-perfect match (95% IoU). This metric reveals how precise and detailed the detections are.

From our experiments, we observed that, except the baseline model, which underperformed due to its smaller input image size, all other models achieved comparable results, with a mAP50 of approximately 0.7 and a mAP50-95 of around 0.3. Interestingly, medium and large YOLO models (model 6-12) did not outperform the small YOLO models. This was likely due to the limited dataset size and training epochs, which led to underfitting in the larger models. Larger models generally require more data and training time to fully leverage their increased complexity. Furthermore, complex models with large batch size such as model 8,9,11, and 12 failed to converge due to memory issues.

We also noticed that dummy classes did not improve model performance (see model 2,4,7). This ineffectiveness of the dummy classes likely stems from the fact that there were no actual labels for the dummy classes in our dataset. As a result, the loss function only computed gradients for the two classes present in the labels, i.e. 'COTS' and 'background', and ignored dummy classes entirely. Additionally, dummy classes were intended to dilute the probability mass across more classes, reducing overconfidence. However, without labels, the logits for dummy classes did not compete with the logits for real classes, so virtually no masses were assigned to dummy classes.

Up to this point, our choice of the best model is Model 1, which offers high accuracy with a reasonable resource consumption. Using the configuration from this model, we inserted CBAM into the first two layers of YOLO small. The result is recorded as Model 5a. Surprisingly, the model did not learn anything (mAP50 consistently close to 0) and the training even terminated early after only 8 epochs. The result was so surprising that we needed to conduct a binary search to find out which overridden hyper-parameters caused the model to degrade substantially. It turned out that Adam Weight decay, cosine learning rate scheduler, and tuned augmentation hyper-parameters were the culprits. We hypothesized that the large gradient updates from AdamW in early stages of training, along with extreme input distortion, disrupted CBAM's learning. As we removed overridden hyper-parameters during the binary search, eventually we ended up with YOLO's default model with CBAM insertion, which is Model 5b in Table 3. The result of this model is pretty comparable to Model 1. There are several possible explanations for why CBAM did not lead to a substantial improvement. First, the model might require more training as CBAM's initial weights were completely randomized as supposed to being pretrained. Second, CBAM's structure might have led the model to rely on the skip connection, effectively ignoring CBAM and resulting in little updates to its weights.

Considering all factors, the best-performing model in our experiments was Model 1, which was YOLOv11 small trained with an input image size of 1280 pixels, a batch

Hyperparameter	Training Value	Default	Reason for Overriding
imgsz	1280	640	Higher resolution images allow for learning of subtle patterns
epochs	20	100	Reduce due to resource constraint.
patience	3	100	Early Stopping Criteria.
cache	True	False	Speed up data loading during training.
workers	2	8	Prevent memory issues caused by data loading.
optimizer	AdamW	auto	Improved weight regularization.
cos_lr	True	False	Use a cosine learning rate scheduler.
close_mosaic	2	10	Disable mosaic augmentation in the last N epochs. Correspond to the reduced number of epochs.
lrf	0.0001	0.01	Final learning rate = 10e-6, allowing for smoother convergence.
warmup_epochs	2	3	Reduce warmup due to the reduced total epochs.
nbs	8 or 16	64	Reduce due to resource constraint.
dropout	0.3	0	Reduce overfitting due to the small COTS dataset size.
degrees	160	0.0	Increase image rotation
translate	0.5	0.1	Increase variability in object positions.
shear	90	0.0	Increase shear augmentation.
flipud	0.1	0.0	Probability of flipping the image left to right.
bgr	0.01	0.0	Probability of flipping image channels from RGB to BGR.
mixup	0.2	0.0	Probability of blending images and labels.

Table 2: Overridden hyper-parameters

Model	Model Size	CBAM	Epochs	Image Size	# Dummy Classes	Batch Size	mAP50	mAP50-95	GPU Usage (Used/Total GB)
Baseline	Small		20	640	0	8	0.31	0.133	2.5/22.5
1	Small		20	1280	0	8	**0.729***	**0.331**	9.1/22.5
2	Small		20	1280	10	8	0.701	0.331	9.1/22.5
3	Small		20	1280	0	16	0.722	0.339	16.5/22.5
4	Small		20	1280	10	16	0.706	0.328	16.5/22.5
5a	Small	Y	20	1280	0	8	0	0	Out-of-Memory
5b	Small	Y	20	1280	0	16	**0.744**†	0.392	17.5/22.5
6	Medium		20	1280	0	16	0.721	0.330	32.5/40.5
7	Medium		20	1280	5	16	0.698	0.321	32.5/40.5
8	Medium		–	1280	0	32	–	–	Out-of-Memory
9	Medium		–	1920	0	32	–	–	Out-of-Memory
10	Large		20	1280	0	16	0.674	0.306	32.5/40.5
11	Large		–	1280	0	32	–	–	Out-of-Memory
12	X-Large		–	1280	0	16	–	–	Out-of-Memory

Table 3: YOLO v11 Fine-Tuning Results
Best Model * | CBAM w/ default settings †

size of 16, and for 20 epochs. This model achieved the highest mAP50 of 0.729 and mAP50-95 of 0.331, while consuming significantly less computational resource than other models. Figure 3 provides the training and validation curves for this model. Both training and validation loss curves show a consistent downward trend, while accuracy metrics demonstrate steady improvement as training progresses. One notable pattern here is the sudden increase in

training losses in the last two epochs, which stems from reducing the hyper-parameter `close_mosaic` from a default value of 10 to 2. However, this adjustment did not negatively impact the model's overall performance. This is evident from the stable trends in validation losses and accuracy metrics, which confirm that the model was not overfitting or underperforming due to the reduction in `close_mosaic`. Instead, the model leveraged its training effectively to generalize to unseen data. Additionally, this model processes images in 7.0 ms per image, comprising 0.4 ms for preprocessing, 5.5 ms for inference, and 1.1 ms for postprocessing. This speed corresponds to approximately 143 frames per second (FPS), making it highly suitable for real-time underwater COTS detection. Figure 2 visualizes the model prediction for a sample frame.

4. Conclusion

In this project, we developed a deep learning model for real-time COTS detection using the state-of-the-art YOLOv11 from Ultralytics and the CSIRO Crown-of-Thorns Starfish Detection dataset. While we experimented with CBAM and dummy classes, these methods did not significantly improve results during the fine-tuning process. However, they might prove effective with increased computational resources, a larger yolo v11 model size, higher input images size and batch size, and more training epochs. Our current best model provides a promising result, with an mAP50 of 0.729, an inference time of 7.0 ms per image (approximately 143 FPS). These performance metrics demonstrate that the model is suitable for real-time underwater COTS detection, improving the efficiency of COTS removal efforts and contributing to the protection of coral reefs.

References

[1] Australian Institute of Marine Science. Reef monitoring sampling methods, n.d. Accessed: 2024-12-07. 1

[2] Jiajun Liu, Brano Kusy, Ross Marchant, Brendan Do, Torsten Merz, Joey Crosswell, Andy Steven, Nic Heaney, Karl von Richter, Lachlan Tychsen-Smith, David Ahmedt-Aristizabal, Mohammad Ali Armin, Geoffrey Carlin, Russ Babcock, Peyman Moghadam, Daniel Smith, Tim Davis, Kemal El Moujahid, Martin Wicke, and Megha Malpani. The csiro crown-of-thorn starfish detection dataset, 2021. 1, 2

[3] Ultralytics Team. Hyperparameter tuning guide. https://docs.ultralytics.com/guides/hyperparameter-tuning/#what-are-hyperparameters, 2024. Accessed: 2024-12-09. 3

[4] Ultralytics. Yolov11: Object detection and image segmentation models. https://docs.ultralytics.com/modes/, 2024. 1, 2

[5] UNESCO World Heritage Centre. Great barrier reef, n.d. Accessed: 2024-12-07. 1

[6] Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. Cbam: Convolutional block attention module, 2018. 1

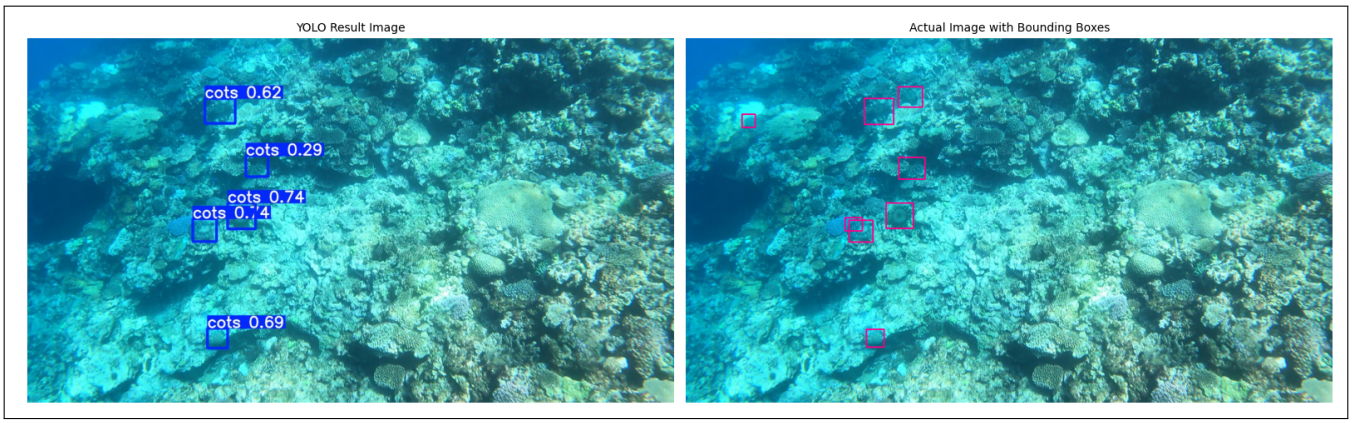

Figure 2: Model Performance Review: Actual vs Prediction with Bounding Boxes and Probabilities

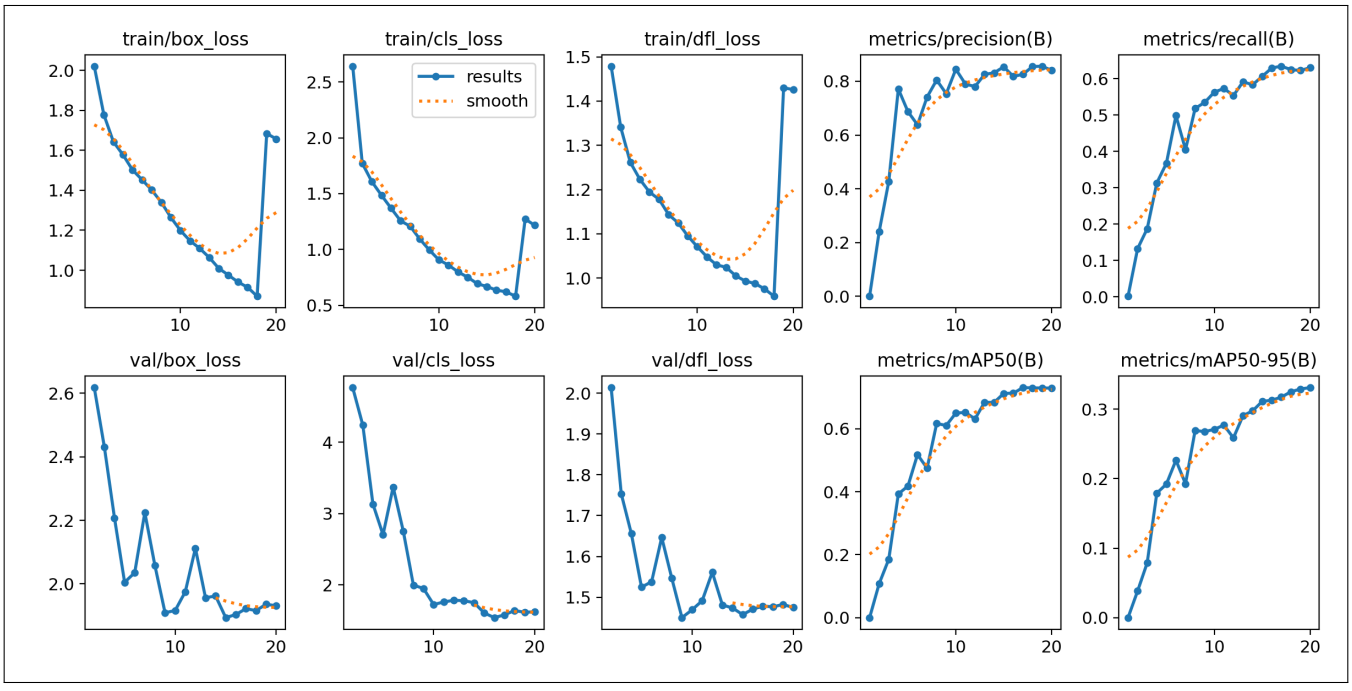

Figure 3: Best Fine-Tuned Model Performance with Multiple Metrics