Detecting COTS in the Great Barrier Reef with YOLOv5

1st Ian Boulis
dept. of Mathematics
Michigan Technological University)
Houghton, MI
isboulis@mtu.edu

2nd Alan Bouwman
dept. of Mathematics and Computer Science
Michigan Technological University
Houghton, MI
akbouwma@mtu.edu

***Abstract*—**
***Index Terms*—component, formatting, style, styling, insert**

I. Background

Australia's Great Barrier Reef is one of the seven wonders of the Natural World. This coral reef not just a beautiful tourist attraction, but also an ecosystem that provides a home to 1,500 species of fish, 400 species of corals, 130 species of sharks, rays, and a massive variety of other sea life [4]. Recently, the Great Barrier Reef has been under threat from the overpopulation of a particular starfish (sea star) – the coral-eating crown-of-thorns starfish (or COTS for short). The Great Barrier Reef Foundation, Australia's national science agency (CSIRO), google, and Kaggle have presented the challenge of leveraging machine learning to come up with a real time detection system for detecting outbursts of these COTS.

II. Motivation

Image detection is at the forefront of the machine learning industry with cutting edge applications such as self-driving cars and advanced robotic systems. Being able to parse in images to a machine that meticulously analyzes it means being able to process a large quantity of data and at a higher detail than humans are capable of. This is incredibly useful for a project such as ours where there are tighter constrains on what a human is capable of doing. Prior to applying machine learning to this problem, a diver used to have to be towed behind a boat and manually survey the area as they're being dragged. This poses a handful of issues such as the diver needing to resurface for air or the diver not being able to catch every detail on the seafloor. Applying a machine learning algorithm such as Yolov5 means being able to detect COTS at a higher and more accurate rate and allow the Great Barrier Reef to survive and prosper.

The Yolo family of models and particularly the Yolov5 model are the cutting edge of image detection/processing. The motivation for us to take on this project was to increase our knowledge of machine learning as a whole and add Yolov5 to our toolbox of machine learning capabilities. Through this project we hope to learn how to implement Yolov5 from start to finish, how to format our data, how to access/load our data, and how to implement and tune Yolov5 via TensorFlow. By the end of this project we hope that our knowledge of the Yolo family of models has improved, along with other aspects of machine learning such as using Roboflow and its API, and even underlying mechanics of how Python operates.

III. Dataset Description

At a high level, the dataset consists of videos taken of the Great Barrier Reef floor. This video data is collected by a method called "Manta Tow", which is performed by a snorkel diver. Any COTS in the video are labeled with bounding boxes. More pratically, the training set consits of a directory called train and a CSV file called train.csv. The train directory consists of .jpg images of the video, labeled with a video id and a frame number. Then in train.csv, each row corresponds to one of the images in the train directory. Each row has the data needed to identity the image (image ID and frame number). Each row also contains a field called sequence, which refers to the ID of the time spent under water and a sequence number (the frame number for that sequence). CSV file also another column called annotations. This column contains the data for all of the bounding boxes of the COTS starfish in that image. The bounding box is given as $((x_{min}, y_{min}), \ell, h)$ where (x_{min}, y_{min}) is the upper left hand coordinate of the bounding box, and ℓ, h are the pixel length and height of the bounding box in that image. There are a total of 23501 images in the training set.

IV. Related Work

V. Overview of Architecture

The Yolov5 model that we use is a single-state object detection system, meaning that it connects the procedure of generating bounding boxes with class labels to create an end to end pipeline that can be optimized much easier. The Yolov5 model consists of a multitude of layers, mostly alternating convolutional neural networks and C3/upsampling layers. Our model can be broken down into three main components: the model backbone, model neck, and model head.

A. Model Backbone

The model backbone is mainly used to extract features from the given image. Yolov5 uses Cross Stage Partial (CSP) networks which is based on DenseNet [3]. These networks are

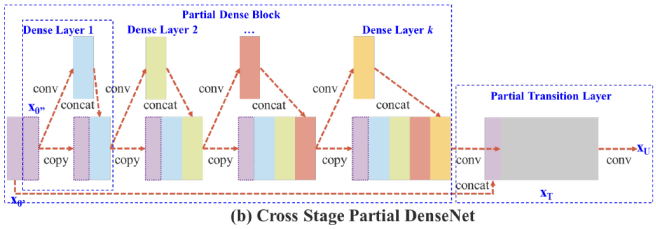

Fig. 1. Illistration of Cross Stage Partial Networks via DenseNet (Source [2])

designed to connect layers in convolutional neural networks which provides a multitude of benefits. First and foremost it helps the model deal with vanishing gradients which occur when using very deep machine learning models, such as ours. This prevents our loss function from trending towards zero and improves model training. CSP also reduces the overall number of network parameters by encouraging our model to increase feature propagation and reuse said features.

B. Model Neck

The model neck is used to generate feature pyramids, which help to generalize image features. This helps the model in the sense that it learns generalized features in any orientation or configuration, and allows the model to perform well on "hidden" or unseen data. This is particularly useful for our dataset as our images are frame by frame videos, so it is very unlikely for two COTS to be in the same orientation. Yolov5 uses Path Aggregation Network (PAnet) to generate feature pyramids [**?**]. PAnet uses bottom-up path augmentation which increases localization signals in lower layers and shortens the path of information between topmost layers and bottom layers. It also implements a technique called adaptive feature pooling, which links feature grids with all feature levels and allows useful information from each layer to propagate directly to subnetworks.

C. Model Head

The model head contains the final layers for Yolov5. The head is responsible for detecting model predictions in the form of confidence scores, size/shape, and accuracy. Our model head consists of several convolutional neural networks that take in the results of the CSPs.

VI. Preprocessing Procedure

The first preprocessing step we had to take was getting our data in a readable format for the Yolo architecture. This proved to be quite the challenge, our data as downloaded from Kaggle required an outside program and API called Roboflow to be usable. The downloaded data was in two separate files, an image directory which contains the several thousand images that the model would take in, and a label directory, which contained a text file corresponding to every image. The text files in this label directory consisted of various rows, a row for each bounding box of our training data. Each row consisted of five numbers, an integer for the class of any object in the photo and since our model is only classifying one class these were all zeros, then the x and y coordinates for each bounding box, along with their length and height. Each text file had the same number of rows as COTS in their corresponding image, and each text file/image contained anywhere between 0 to 10 COTS.

Having information spread out between several thousand files and between two folders was a very foreign concept to us, and we were able to used Roboflow to alleviate this. Roboflow is an online application where users can upload image data in various formats, and luckily for us there is a system in place that matched the format of our data. Using Roboflow we uploaded our images and labels separately and used their website to automatically match annotations to their image and add bounding boxes to the respective image. After creating bounding boxes for each image based off annotations, Roboflow allows us to apply preprocessing before uploading the data. We chose to go with their default preprocessing as anything beyond this is a paid for service. The default preprocessing consists of auto-orientating the images and stretching/compressing the images to a default pixel size, in our case 416x416. Afterwords we upload our data as a public dataset and can access our data in Google Colab.

Accessing our dataset in Colab required the use of Roboflow's API, which posed a handful of challenges. Surprisingly enough, Roboflow's API is easy to understand once you've access your data through it once. To access your data it requires several lines of code, one to establish your API key, one to access your 'workspace' which is a directory of your different datasets, one to access your project which is a specific dataset within that directory, and finally one to access the version of data you want. This sounds intuitive enough, but for some reason we were having issues accessing the version of our data that we needed. After reloading the workspace about a day after uploading the data we were able to access our specific version and continue with model building. My assumption for this issue is that the API needed time for the dataset to fully upload and be able to access it.

VII. Training

A. Data Augmentation

Outside of preprocessing, Yolo performs several types of data augmentation while in the train phase which are just as important for the end performance of our system. Yolo performs standard data augmentation such as scaling, color space adjustments, but most importantly mosaic augmentation. The augmentation settings were as follows: Translation factor of up to 0.1, scale of up to 0.5, and probability of 0.5 of a left-right flip. We also vary the Hue of the color space by 0.015, the Saturation of the color space by up to 0.7, and the Value of the color space by up to 0.4 (for HSV).

Most notable is the use of masaic augmentation. Mosaic augmentation was first introduced in Yolov4, and is present in Yolov5, our model. Mosaic augmentation combines a series of 4 images into a tile of random ratios to create a new image, which the model then learns on [**?**]. This is an important step

in our model specifically, as mosaic augmentation allows the model to learn to identify objects on a smaller scale than normal. Since our data has images taken at various distances from the sea floor, some COTS appear larger than others and in different positions. Mosaic augmentation is a crucial step in ensuring our model does not over-fit on our training data.

B. Hyper Parameters

We used the Hyper-parameters provided by Yolo for low data augmentation training. The learning rate was set to 0.01, momentum of 0.937, weight decay 0.0005. The optimizer used was stochastic gradient decent (SGD).

C. Training Metrics

We record several metrics throughout training to determine if our model converging. We can also use these metrics to help detect over-fitting. The metrics that we kept track of throught each epoch are the box loss, object loss. The precision, recall, and mean average precision (mAP) are also calculated at the end of each epoch on the validation data only. The Yolov5 architecture also keeps track of class loss, but since there was only one class to predict in this problem. the class loss was always zero. The box loss for the Yolov5 architecture is computes as follows [7]:

$$L_{\text{Box}} = \sum_{i=0}^{s^2} \sum_{j=0}^{B} I_{i,j}^{\text{obj}} \left[1 - GIoU\right]. \tag{1}$$

Here, s^2 is the number of grids and B is the number of bounding boxes in each grid. Also, $I_{i,j}^{\text{obj}} = 1$ when an object exists in a bounding box and is 0 otherwise. $GIoU$ the the Generalized Intersection over Union. The $GIoU$ is computed as follows: Let A be the bounding box predicted by the model, and B be the ground truth bounding box, and let C be the smallest rectangular region covering both A and B. Then,

$$GIoU = \frac{|A \cap B|}{A \cup B} - \frac{|C \backslash (A \cup B)|}{|C|} \tag{2}$$

(see figure 2).

The object loss is related to the confidence of the prediction and can be calculated as follows:

$$L_{\text{conf}} = -\sum_{i=0}^{s^2} \sum_{j=0}^{B} I_{i,j}^{\text{obj}} \left[\hat{C}_i^j \log(C_i^j) + (1 - \hat{C}_i^j) \log(1 - C_i^j)\right]$$
$$- \lambda_{\text{noobj}} \sum_{i=0}^{s^2} \sum_{j=0}^{B} I_{i,j}^{\text{noobj}} \left[\hat{C}_i^j \log(C_i^j) + (1 - \hat{C}_i^j) \log(1 - C_i^j)\right].$$

Here, $\hat{C}_i^j$ is the predicted confidence of the jth bounding box in the ith grid and C_i^j is the true confidence.

We also calculate the precision R and recall R for the model on the validation data. Recall that

$$P = \frac{TP}{TP + FP},$$
$$R = \frac{TP}{TP + FN}.$$

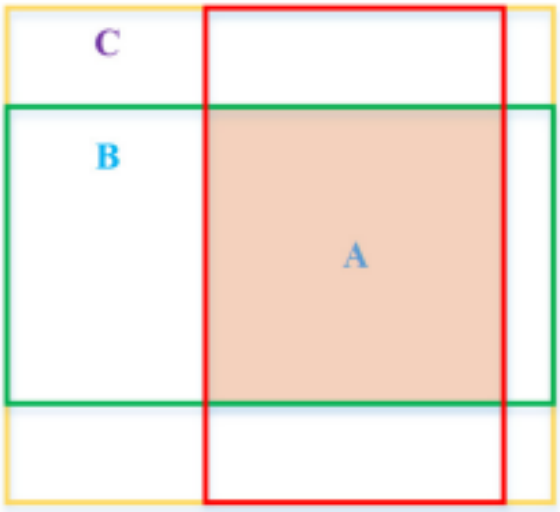

Fig. 2. Calculation of GioU (Source [7])

A bounding box is predicted as a True Positive (TP) if it IoU with that box and the ground truth is greater than or equal to a certain threshold, $\kappa \in [0, 1]$, (by default, $\kappa = 0.5$). otherwise, that bounding box is predicted as a False Positive (FP). False Negatives (FN) occur when there is a ground truth bounding box without a predicted bounding box with an IOU of greater than or equal to κ.

These metrics bring us to the Mean Average Precision mAP, which can be calculated as follows:

$$mAP[m : d : M] = \frac{1}{C} \sum_{\kappa} P(\kappa) \Delta R(\kappa). \tag{3}$$

Here C is the number of classes and κ ranges between $m \leq \kappa \leq M$ with a step size d. For our project, $C = 1$ since we are only detecting COTS star fish. Unless otherwise specified, mAP will denote mAP only taken at $\kappa = 0.5$.

The box loss curve looks very good. The model seems to converge around epoch 130 to a validation box loss of about 0.23. In the epochs from 130 to 150 the training box loss still falls slightly from about 0.27 to about 0.25. The indicates that the model may be slightly overfit (see figure 3).

The object loss curve also looks promising. Interestingly, around epoch 30, the model seems to perform quite a bit worse on object loss in both training and validation, and then it corrects itself. The same behaviors can be seen in the precision and recall curves, along with the mAP curves. Like the box loss, the object loss on the validation data also seems to stabilize around epoch 130 to about 0.006. The object loss for the training data is also roughly stable throughout these epochs. The object loss may not suffer from as much over-fitting as the box loss.

The model also achieves very high precision and recall on the validation data after 150 training epochs. The model achieves its best precision in epoch 147 with a precision of 0.956. Thus, over 95% of the bounding box predictions made on the test data were correct. The best recall happens in epoch

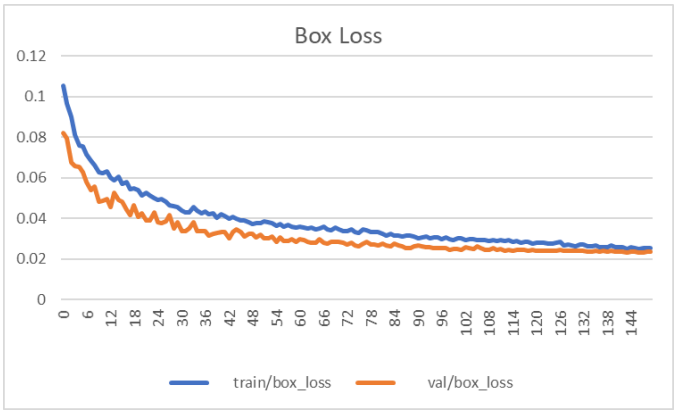

Fig. 3. Box Loss

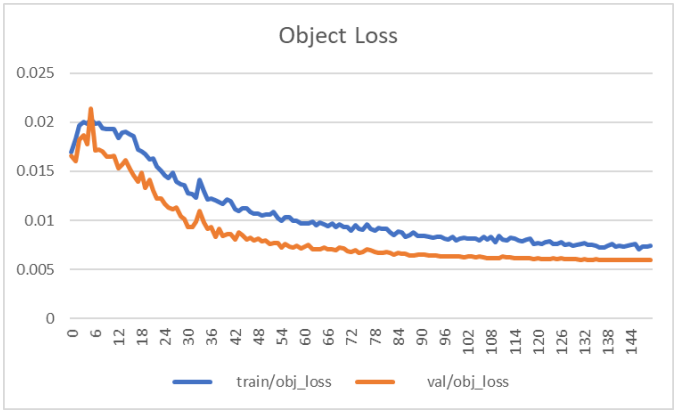

Fig. 4. Object Loss

143 with a recall of 0.947. That is, of all of the truth bounding boxes, our model predicts almost 95% of them correctly. See figure 5.

The mean average precision also looks very promising. Hoever, the mAP[0.5:0.05:0.95] is relatively low, achieving a maximum of only about 0.55 and appearing to stabilize around here. This means that Yolov5 is doing much worse

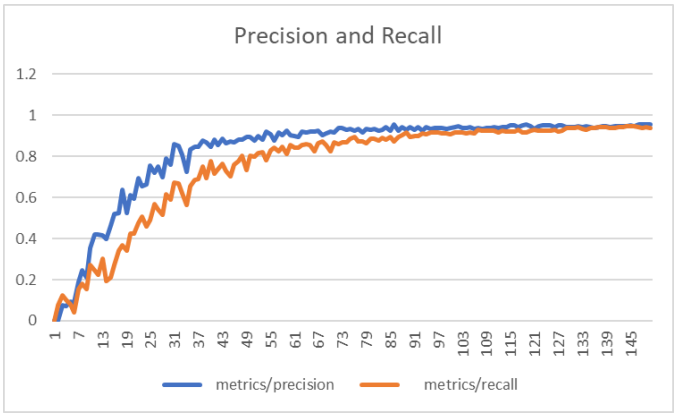

Fig. 5. Precision and Recall on validation data vs Epochs

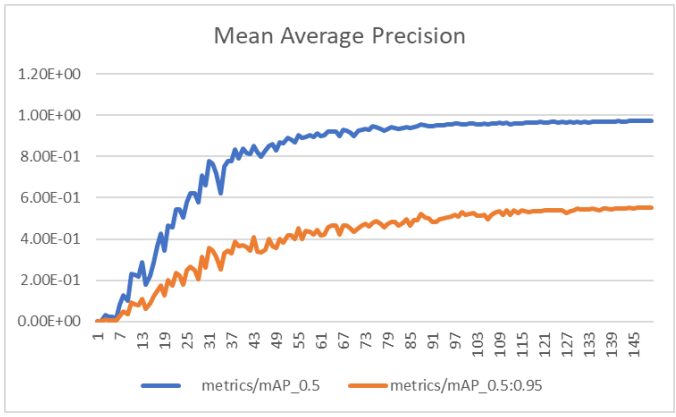

Fig. 6. mAP score on validation data vs Epoch

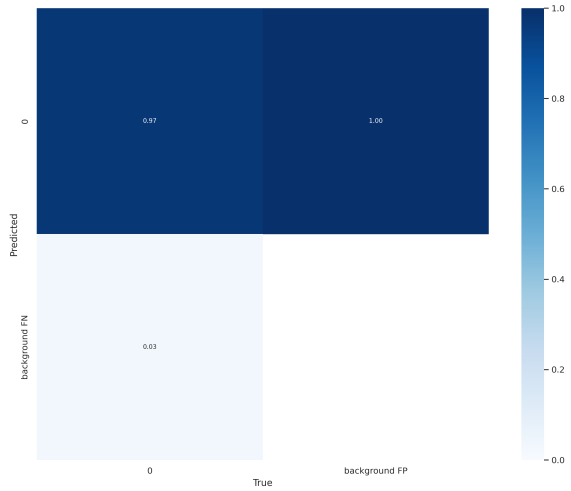

Fig. 7. Confusion Matrix

at predicting boxes within 95%IoU of the ground truth (see figure 6).

VIII. Experimental Results

The results of our experiment indicate that our model learned well on our training data and was able to generalize to perform well the validation data. First we look to the confusion matrix (figure 7). The confusion matrix indicates there are a lot of false positive detected. It also indicates a lot of success on True Positives 0.97. The confusion matrix indicates very low prediction of false negatives 0.03, meaning that our object detector is missing almost none of the ground truth boxes. True Negatives are not very meaningful in the task of object detection, so a score of 0 is given by default.

We next turn our attention to the P curve. The P curve (figure 8) shows the precision of the object detector for different levels of confidence thresholds on the bounding boxes. So for a very low confidence threshold, the object detector has very low precision because almost all of the detections are false positives. The figure is telling us that at a confidence threshold 0.932 the detector performs with perfect precision (all detected

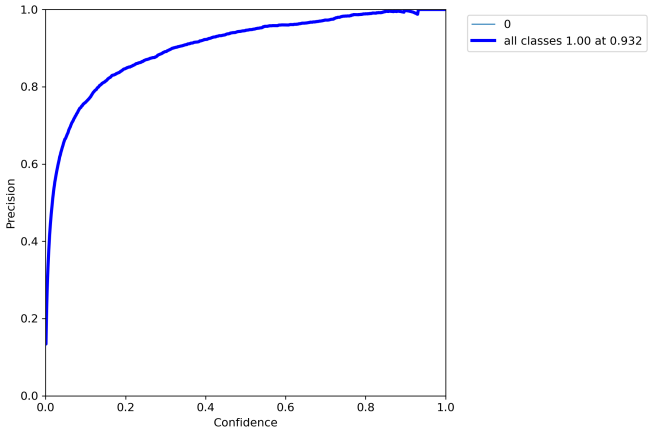

Fig. 8. Precision Curve

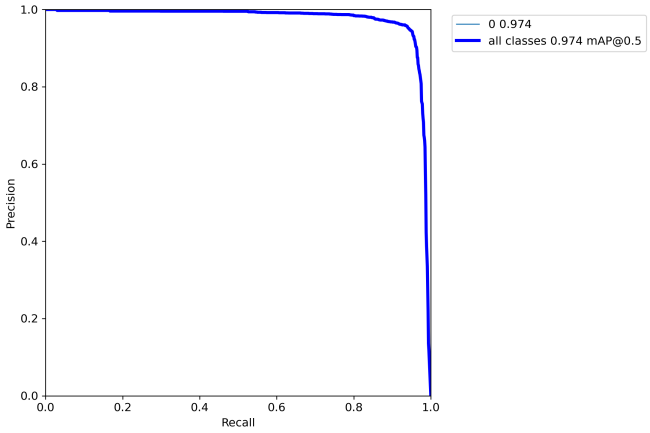

Fig. 10. PR Curve

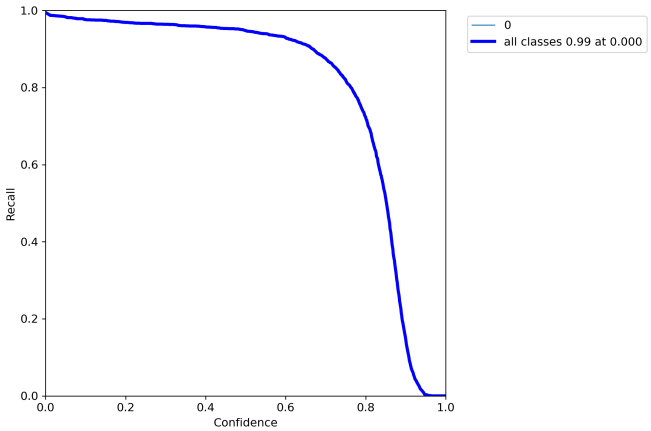

Fig. 9. Recall Curve

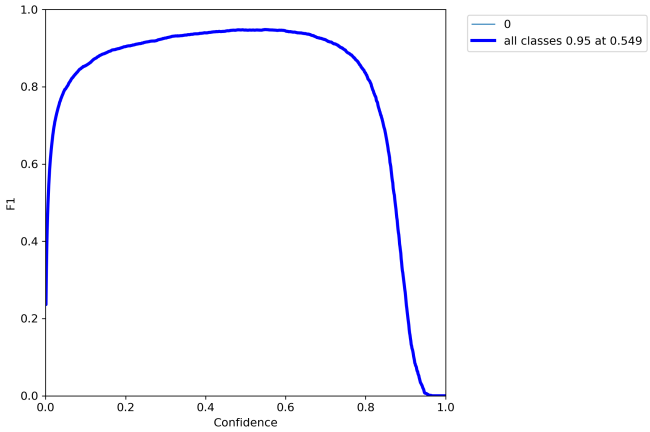

Fig. 11. F_1 Curve

positives are true positives). This is quite a bit higher than the default confidence threshold needed for detection of 0.5.

The P curve fails to tells us certain information, like how many objects are we failing to detect when we insist on an extremely high confidence? To gauge this, we can look to the recall curve (figure 9). Like the precision curve, this plots the recall at different confidence thresholds. When the confidence threshold is low, recall is very high because there is no room to make type ii errors. When the confidence threshold is very high, only type ii error occurs.

We also turn our attention to metrics that balance precision and recall. The PR curve plots both the precision and the recall of the detector at different levels of confidence thresholds. Our PR curve (figure 10) looks consistent with a model that has learned well. At a confidence threshold of 1, all positives detected are true positives (so the precision is 1), but it could be the case that almost no objects are actually detected (so there are many false negatives) which pushes the recall to 0 (top left of the curve). At the bottom right of the curve, the confidence threshold is zero so almost all of detection are false positives, pushing precision to zero. However, there is no room for making false negative errors, so the recall is 1. The AUC (area under the curve) of the PR curve is a good measure that balances both precision and recall. The AUC of this curve is 0.947. This is a pretty good AUC and it indicates that our model is balancing precision and recall quite well.

Another metric that balances both precision and recall is the F_1 score. F_1 is calculated as

$$F_1 = 2\frac{PR}{P+R}. \tag{4}$$

The F_1 curve shows the F_1 score for different confidence thresholds. Again, when the confidence threshold is close to 0 the precision will be close to 0, and thus the F_1 score will be as well. On the other hand, when the confidence threshold is close to 1 the recall will be close to 0 and thus the F_1 score will also be close to 0. Our F_1 curve (figure 11) is consistent with this behavior. We also note that the F_1 score is maximized at a confidence threshold of 0.549 on the validation data. This indicates that at test and performance time, we should set the confidence threshold to about 0.55 to ensure the best performance. This is very close to the default confidence threshold of 0.5.

IX. Conclusion

A. Limitations

One of the main limitations of our project is the lack of results on test data. Although Yolov5 provides great means of looking over and analyzing results from training and validation data, they provide no means of analyzing results on test data. Out of the box, the only thing which Yolov5 provides is detection. Obtaining metrics like mAP and F1 are not possible without serious modification of the source code.

Another limitation is that we never got a chance to consider more models. It would have been interesting to see if older versions of Yolo, like Yolov3 would actually perform better on this specific data. It may have also been interesting to try some other models such as efficient det.

Another limitation is that we never met our goal of using object tracking. By nature, there is a lot of time dependence in a lot of our data (between images). Yolov5 does not explicitly take advantage of this. In fact, training and testing images are shuffled out of order. It would have been interesting to see if adding an object tracking algorithm, such as DEEPSort [5] could have improved performance.

B. Strengths

Despite our project's limitations, we were still able to make a model that performed very well on the training and validation data. Using an object tracking algorithm, such as DEEPSort could have improved performance, but it also could have potentially been overkill. Through doing this project, we were able to learn a lot about the theory of object detection, including common architectures, how they work, and object detection performance metrics. We also gained practical experience in implementing object detection on a state of the art detector while integrating API's such as roboflow. On a final our object detector is certainly powerful enough and fast enough to be used to detect COTS starfish in the Great Barrier Reef.

Acknowledgment

We would like to acknowledge our professor, Xiaoyong (Brian) Yuan, for teaching us a great deal about machine learning and convolutions neural networks, and for the opportunity he presented to us to work on these projects.

References

[1]
[2] Chien-Yao Wang, Hong-Yuan Mark Liao, I-Hau Yeh, Yueh-Hua Wu, Ping-Yang Chen, Jun-Wei Hsieh, "CSPNet: A New Backbone that can Enhance Learning Capability of CNN". https://arxiv.org/abs/1911.11929 Parvinder Kaur; Baljit Singh Khehra; Er. Bhupinder Singh Mavi, "Data Augmentation for Object Detection: A Review". https://ieeexplore.ieee.org/abstract/document/9531849.
[3] P. Kaur, B. S. Khehra and E. B. S. Mavi, "Data Augmentation for Object Detection: A Review," 2021 IEEE International Midwest Symposium on Circuits and Systems (MWSCAS), 2021, pp. 537-543, doi: 10.1109/MWSCAS47672.2021.9531849..
[4] "Help Protect the Great Barrier Reef". Kaggle.
[5] S. Kumar, Vishal, P. Sharma and N. Pal, "Object tracking and counting in a zone using YOLOv4, DeepSORT and TensorFlow," 2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS), 2021, pp. 1017-1022, doi: 10.1109/ICAIS50930.2021.9395971.
[6] yolov5 https://github.com/ultralytics/yolov5
[7] Qisong Song, Shaobo Li, Qiang Bai, Jing Yang, Xingxing Zhang, Zhiang Li, and Zhongjing Duan, "Object Detection Method for Grasping Robot Based on Improved YOLOv5". *Micromechanics*. September, 2021.
[8] "F-Score" Wikipeida.
[9] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, Jiaya Jia, "Path Aggregation Network for Instance Segmentation". https://arxiv.org/abs/1803.01534#.