Detecting the Multiple States of Oyster Activity and Orientation using Deep Learning Image Processing and Computer Vision Algorithms

Joshua Comfort
Department of Computer Science, Salisbury University
jcomfort1@gullnet.salisbury.edu

Ian Rudy
Susquehanna University
rudyi@susqu.edu

Dr. Yuanwei Jin, UMES
Dr. Enyue (Annie) Lu, Salisbury University

***Abstract*— The United Nations projects that the global human population will grow to eleven billion by 2100. As the populace increases, the demand for food will likewise grow. The global farming sector will need to help meet this increased need by boosting its efficiency and production volume. These industries will need to increase their output by improving their current practices by implementing innovative technologies that improve growth. One area, in particular, needs much improvement, the oyster farming industry still uses practices from the 19th century. Modernizing oyster farming has the potential to provide large quantities of a high-protein sustainable food source. This research aims to create an automated monitoring system that will allow oyster farmers to track their oyster crops' health and activity remotely. To create such a system, we utilize high-performance computing and deep learning to adapt an object detection model, YOLOv5_OBB, to recognize oysters under three different states of activity. By periodically using the object detector, farmers can use the activity to help infer the health of their oyster crops, reducing the amount of work required and thus increasing efficiency. In addition to applications in aquaculture, deploying the systems developed in this project can benefit oyster restoration efforts. For example, helping monitor the health of the restored populations, such as those in the Chesapeake Bay.**

***Index Terms*—Detection, Oysters, Aquaculture, Food, Orientation, GPU, HPC, Activity, YOLOv5, USDA, Industry**

I. Introduction

The United Nations projects that the global human population will increase by two billion people by 2050, leveling off at over 11 billion people in 2100 [1]. As the population grows, there is a forecasted influx in the global food demand for food ranging anywhere from 60-90%. Currently, the preparation of the international food production sector is inadequate to deal with such an enormous surge in demand [2, 3]. To be equipped to match this new stipulation for food requires the creation of entirely new industries. The existing areas need to implement modern practices and increase efficiency using modern technologies. The shellfish farming industry is one sector with the potential to meet the increase in food demand. Shellfish are both a sustainable and environmentally friendly source of high-protein food; the industry also provides opportunities for economic growth along coastal regions [4].

However, the current practices within the aquaculture industry lack the technological advancements in the rest of the agricultural industry. In particular, the oyster farming sector still utilizes the same practices established during the 19th century, which are both inefficient and labor-intensive [5]. With the oyster industry in desperate need of updated practices, it is quickly falling behind the rest of the food production sector [6]. Considering the current limitations, the Food and Agriculture Organization of the United Nations ranks the industry as having the potential to experience massive growth [7]. Therefore, this research will explore one of the many possible technological advancements within the field. This project will study the prospect of using automated monitoring systems to increase efficiency by utilizing deep learning algorithms. By using monitoring systems, farms can use the oyster crop's activity to infer the health of the population and their adaptation to the environment.

In this research, we adapt and optimize a deep learning model capable of detecting oysters' activity and orientation. By classifying oysters into three states of activity, closed, semi-open, and open, the model will be capable of monitoring oysters over time. In addition, this project explored the possibility of using each oyster's orientation to increase the accuracy of predictions. Previous research groups have sought to implement similar systems; however, this project improved the previous work by using orientation to provide more information about each oyster and generalizing the model to work across multiple environments [8]. In addition to its applications within the aquaculture industry, the use of this project's advancements may prove beneficial to oyster restoration endeavors. For example, remotely monitoring the oysters will help reduce the labor required across many locations, such as the work done within the Chesapeake Bay [9].

II. Related Work

Within the confines of this field of research, there have been various attempts at creating an oyster detection system. The first of these systems reviewed was a class project by an undergraduate student at the University of Maryland Eastern Shore (UMES) [10]. The student was tasked with creating a fish classification algorithm to classify fish images

The work is funded by NSF CNS-2149591 under Research Experiences for Undergraduates (REU) Program and by USDA National Institute of Food and Agriculture.

based on the species they contained. The model created could classify five different fish species with accuracy ranging from 70-96%, depending on which species was being classified. To begin this research, we started by reimplementing this project using oysters to learn the basics of machine learning and better understand the tools being used. However, this system was only an image classifier and did not localize the oysters within the image; thus, it was left behind in search of more applicable models [11].

The second piece of material reviewed was a comprehensive capstone design project done by students at UMES [12]. The project covered from design to an implementation capable of running an object detection system on a RaspberryPi. The model used by the researchers could localize oysters under specific conditions and classify their activity. The project took advantage of existing convolutional neural networks to create the oyster detection system, therefore requiring less direct implementation [13]. However, the model's performance may be limited in real-world applications due to the dataset being composed of images from a tank of oyster shells placed in different activity states, possibly limiting the model's performance in underwater environments [14].

The final related work review was a research paper published by researchers at the University of Maryland College Park [15]. The project utilized a Remotely Operated underwater Vehicle (ROV) to take pictures of oysters in various underwater locations. The annotated images were fed into Facebook Research Group's Detectron2 model. As a result, the researchers were able to train an accurate model capable of localizing the oysters within an image. However, this research did not seek to classify the oyster's state of activity. Additionally, the model's accuracy was around 79%, thus leaving room for some improvement [16]. Consequently, our project takes the strengths of each of these projects and combines them into a single general model capable of oyster activity and orientation identification.

III. Methods

In order to detect an oyster and its orientation within an image, an object detector needs to accomplish three steps, classification, localization, and bounding box rotation. Combining all of these steps, one is able to implement a fully functioning system that can identify oysters' orientation.

A. *Classification*

The Convolutional Neural Network (CNN) is the most common type of classifier. The CNN is a particular type of neural network that seeks to mimic the way the human visual cortex works by simulating virtual neurons. Each of these neurons contains a mathematical function that is only activated if the value input passes a defined threshold. By connecting these neurons, they can be made to simulate the behavior of a human brain [17]. As a result, a CNN can reduce the dimensions of an image without losing the information contained within, thus reducing the amount of computational power needed to classify it. To achieve its classification, a CNN uses three main layers, convolutional layers, pooling layers, and a fully connected layer. The convolutional layer works by passing a filter over each color channel of the input image; these values are then summed and output into a feature map.

FIGURE I

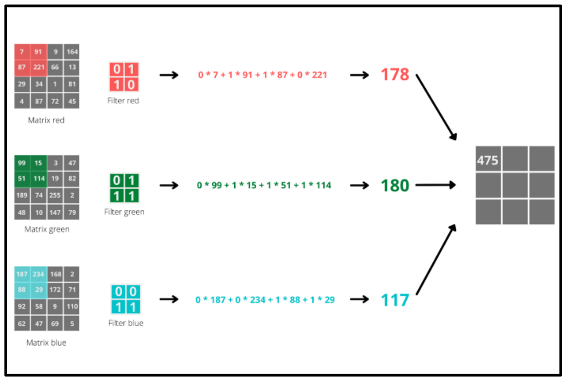

Fig 1. This figure shows a convolutional layer with 2x2 filters, each filter passes over a layer of the input image, and their results are summed into the feature map [14].

Each filter has specialized weights to look for specific features within the input image. The programmer may manually specify the weights; however, much more commonly, weights are learned by the network by feeding it the data to be classified in a network training process. By looking for specific features, the network can save the dominant features that make up the image while reducing the dimensionally [18].

The output of the convolutional layer is passed into the next section of the model, the pooling layer. The pooling layer is designed to shrink the dimensions of the feature map further while still extracting the details. Like the convolutional layer, the pooling layer uses a filter to pass over the feature map and extract information. There are two kinds of pooling, max, and average pooling. Max pooling takes the maximum value from the filter and outputs that value into the reduced feature map. In contrast, average pooling takes the average value of the filter and places that value into a reduced feature map.

FIGURE II

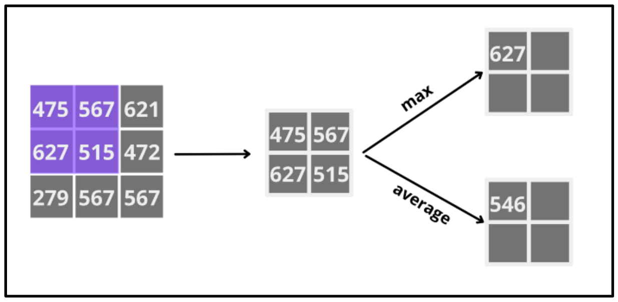

Fig 2. The above figure shows the two kinds of pooling. Max pooling takes the maximum value, and average pooling takes the average of the values [15].

In addition to reducing the dimensionality, the pooling layer also reduces noise within the image. An example might be removing the background from the subject or filtering out other objects that are not being classified [19].

Lastly, the output of the convolutional and pooling layers is passed to the fully connected layer. The fully connected layer contains the neurons that will be used to classify the image. These neurons learn which of the features extracted make up each object being classified. During the training process, the neurons' weights in this layer are updated to match the training data. The output of this layer is a class label with the value that the model thinks the image contains [20]. After the network is trained, it can be used to classify images it has not seen before. Together, the layers make up a CNN model, and these models can be of different sizes or depths depending on the application.

B. Localization

Image localization builds upon image classification; usually, the first layers of an image localizer are made up of CNN layers. However, unlike a convolutional neural network, an object localizer predicts four values, the x and y coordinate, and the width and height rather than a prediction label. The four output values draw a bounding box around each object of interest. In contrast to object detection, localization, like classification, traditionally only works for a single object within an image. Object detection combines localization and classification for multiple objects within an image [21].

C. Detection

YOLOv5 is a single-stage object detector made of three major components, the backbone, neck, and head. Similar to the convolutional layers of a CNN, the backbone's primary purpose is to extract key features from the input image. This project used the YOLOv5 Oriented Bounding Boxes (YOLOv5_OBB) based on the YOLOv5 architecture. YOLOv5 (You only look once) is built on a system that divides the input image into a grid. Each region is accountable for detecting the objects inside itself within the grid system [22]. YOLOv5 uses a backbone known as DarkNet-53, which provides better performance than previously used models. A traditional CNN can be transformed into a model backbone by cutting off the network's final layers and replacing them with filters that will predict bounding boxes. The neck then generates feature pyramids which the model uses to generalize objects at different scales. Pyramids help the model learn the features of objects it is trying to detect, allowing for generalization. YOLOv5 takes advantage of a Path Aggregation Network, also known as PANet, which attempts to boost the information flow by shortening the path it needs to follow. Finally, the model head performs the actual detections. The head draws the bounding boxes around objects and produces the classifications [23, 24, 25].

FIGURE III

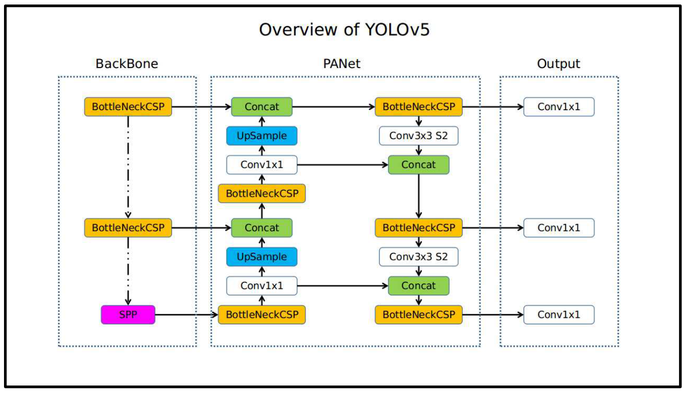

Fig 3. The above figure shows an illustration of the YOLOv5 architecture. The backbone feeds into the feature pyramid network, which then passes its information to the head. Illustration obtained from [26].

D. Orientation

YOLOv5_OBB builds upon YOLOv5 further by allowing for more accurate bounding boxes. The boxes produced by YOLOv5_OBB have an additional factor, orientation. Oriented bounding boxes allow the model's output to fit the objects of interest much more closely and provide additional information about the object. By building off YOLOv5, the oriented model keeps the same architecture but splits off the original model's head to add orientation to the bounding boxes.

FIGURE IV

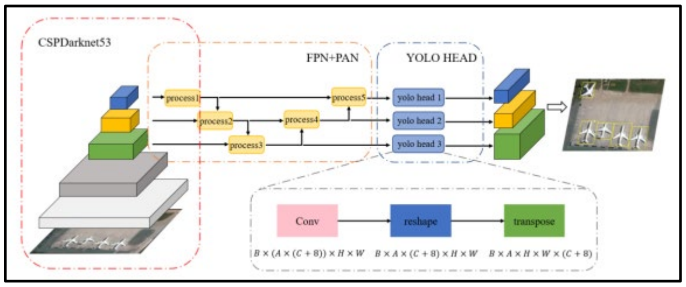
CSPDarknet53
FPN+PAN
YOLO HEAD
process1
process2
process3
process4
process5
yolo head 1
yolo head 2
yolo head 3
Conv
reshape
transpose
$B \times (A \times (C+8)) \times H \times W$
$B \times A \times (C+8) \times H \times W$
$B \times A \times H \times W \times (C+8)$

Fig 4. This figure depicts YOLOv5_OBB architecture, similar to YOLOv5, and the significant difference is the orientation calculator splitting off of the model's head. This image was obtained from [23].

Some orientation detectors [27] opt for an eight-parameter representation for rotated bounding boxes, storing the x and y coordinates for each of the four vertices. YOLOv5_OBB, on the other hand, represents the boxes using five different values: the center x value, the center y value, the box width of the box, the height of the box, and the angle at which it is rotated [28].

FIGURE V

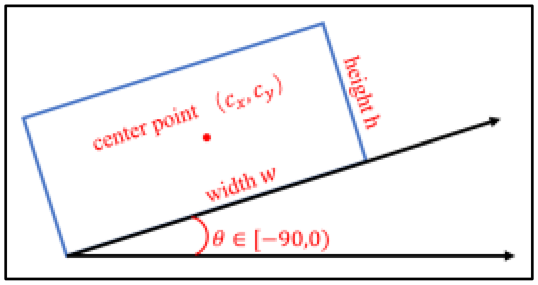

Fig 5. The above image illustrates how YOLOv5_OBB represents the rotation of bounding boxes. Obtained from [28].

This project builds upon YOLOv5_OBB by providing additional information about the detected oysters using the information provided for each bounding box. By using the angle theta, length, and width, arrows parallel to the orientation axis are drawn. However, the direct orientation cannot be inferred due to the lack of information about the oyster's contours within a given image. Some viable solutions to this problem can be seen in the future work section. In addition to the arrows drawn, the oyster's orientation is calculated using the ratio of length to width and the area of the box.

$$weighted\ rot\ factor = \frac{rotation\_factor * (width * height)}{max(width, height) * 0.01} \quad (1)$$

The above equation gives a good approximation of the rotation of each oyster; the rotation factor is calculated by taking the ratio of the shorter side over the long side. Then, multiplying by the area of the bounding box, the scale of each oyster is taken into account. In addition, the rotation factor is scaled by dividing by 1% of the longest side. Thus, the smaller the rotation factor, the more of the oyster's face is visible to the camera; therefore, the greater confidence the model can have in its predictions.

IV. Problems and Challenges

During this research, multiple complex challenges needed to be addressed before a final automatic detector could be produced.

A. *Lack of a Dataset*

The first challenge was creating a dataset, which was a challenging task. Traditionally, underwater datasets have been both expensive and time-consuming to compile due to the nature of underwater environments [29]. However, after searching, it became apparent that no existing oyster datasets were publicly available. Thus, this project required the creation of a dataset containing oysters within various environments. Unfortunately, finding good-quality images proved quite challenging. Therefore, the target size for the dataset was set at 1000 images, allowing for a suitable generalization of oysters while not requiring the majority of the research time. In addition, selecting images with various angles, subject sizes, and water clarity for the training set will allow the model to work in many real-world situations.

Initially, around 225 images of oysters were provided; however, each image needed labeling to fit the needs of this research [30]. Due to the nature and scale of this research, this project did not have the equipment or time to collect real-world images; thus, the remaining images for the dataset were collected from online sources. Each image needed to be annotated, manually cropping each oyster using polygons. All of the annotation for this project was done in Roboflow, an online dataset creation platform that allows for expiration into various object detection formats [31]. The dataset split the oysters into three states of activity: open, semi-open, and open. While there was no absolute metric, the labeling of the oysters followed the guidelines established in the Oyster Activity Detection System Report at UMES [32]. Closed oysters' openings range from 0-0.4 cm, semi-open comprised oysters from 0.5-0.9 cm, and open oysters are those with an opening greater than 1 cm.

By splitting the oysters into distinct states of activity, the oyster crop's activity can be observed by taking inference on frames over time. Each state of activity is separated into a different class, represented by a unique color allowing quick reference without needing to read the label.

FIGURE VI

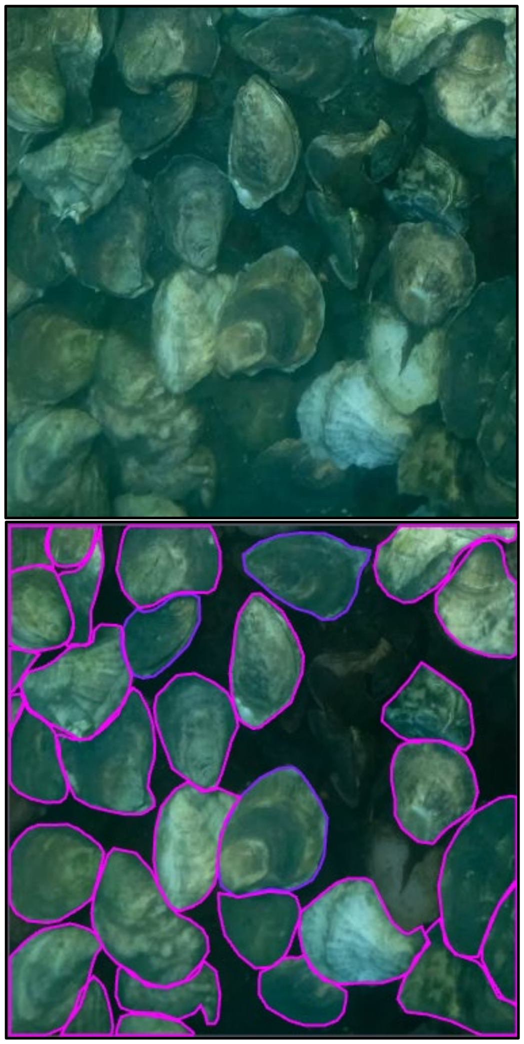

Fig 6. An example of images of oysters before and after annotation. Different oyster activities are differentiated by color. For example, closed oysters are shown in magenta, and semi-open oysters are dark blue.

Additionally, data augmentations were applied to the dataset to increase the variety of information. Data augmentations are filters applied to the input images that change the data to represent different real-world scenarios. For example, crops and zooms stand in place of different camera locations, sheers and rotations show different camera rotations, and brightness and contrast mimic different lighting scenarios [33]. The augmentations on the training set help the model recognize the oysters under adverse conditions such as low light conditions, cloudy water, and varying types of obstructions.

FIGURE VII

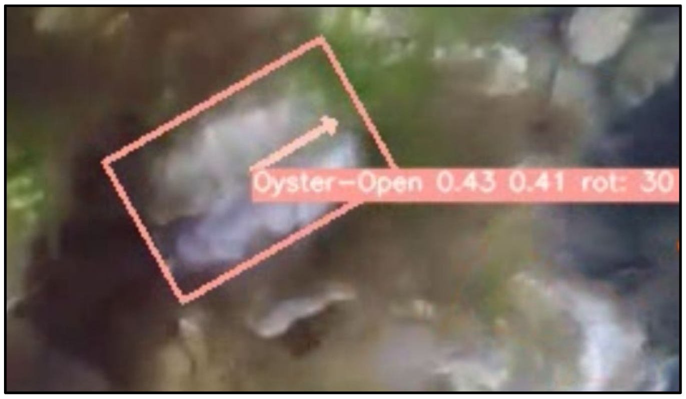

Fig 7. The above example is the YOLOv5 object detector detecting an obscured oyster in slightly cloudy water.

Furthermore, by selecting the data from numerous different sources, the dataset helps ensure that model covers as many environments as possible. A diverse dataset improves over previous implementations that used relatively homogeneous data [34].

B. *Complexity in Training*

The second challenge addressed was training the object detection models using the dataset. Before training an object detection model, the dataset is split into three segments, training, validation, and testing data. The training data is used to teach the model, while validation and testing are used to evaluate the model's performance. The dataset is split into batches, then fed to the model in steps called epochs. The model's performance is improved by adjusting the number of epochs, batch size, and image size during the training process. This project used 640px by 640px images with varying batch sizes. The number of epochs varied from run to run but usually ranged from 250 to 1000.

All object detectors were programmed in Python, a relatively computationally demanding language compared to C and C++ [35]. Due to the scale and complexity of the detector architectures, the amount of time and computational power needed to train the models were not readily accessible [36]. To compensate for the lack of high-powered computers, the object detectors were trained using Google Colab, Google's virtual remote Python environment [37]. Colab provides users access to High-Performance Computing powered by GPU acceleration accessible through a web browser. Nevertheless, even with access to high-performance GPUs and high RAM runtimes, the training process still took multiple days to run each time. Thus, a significant amount of time was required for the training process, meaning that improving model performance through iteration was quite tricky. Multiple training instances were run in parallel using several computers to remedy this.

C. *Orientation Identification*

The final major issue tackled was identifying the oysters' orientation. Again, the YOLOv5_OBB model proved a valuable tool for accomplishing this task. In addition to the oriented bounding boxes provided by the object detector, arrows and rotation values were added. The arrows point parallel to the orientation axis and are color-coded based on the orientation of the rotation factor of the oyster. The rotation factor is calculated using the ratio of the short side of the bounding box over the long side. This information can fully detect the oysters' orientation and create a more accurate detection model. Although such a system could not be implemented due to time constraints, those wishing to continue this research should visit the Github repository to find more information.

V. Procedure and Experimentation

In this project, we started by repeating the fish species classification project at UMES to understand the systems and tools. The project was reimplemented using oyster images rather than previously used fish species. After the reimplementation, more research was conducted to learn more about the direction to take in this research. As stated previously, it became apparent that there were no available datasets of oyster images, so one would need to be created for this project. Additionally, after reading the documentation provided and speaking with the faculty mentor, an oyster object detector was decided upon. Initially, as images were collected, they were labeled into one of two states, open or closed. After around 350 images had been collected, the research set out to implement object detection using this data.

Due to the time constraints and wealth of preexisting models, this research took advantage of available object detection models for implementation. The first model used was Facebook Research Group's *Detectron2*. *Detectron2* is a deep learning model specializing in image segmentation and object detection [38]. The research group also provides access to the model zoo, a collection of model backbones. The three backbones trained were Faster_RCNN_R_50_C4_3x, Faster_RCNN_X_101_ 32x8d_FPN_3x, and Retinanet_R_101_FPN_3x. Each model provides its benefits and disadvantages; thus, robust training was done on each backbone [39]. However, due to some constraints of the *Detectron2* model and a shift in the project's goals, this model was left behind in search of another architecture.

Next, we focused on working with the *YOLOv5* object detector. Like *Detectron2*, YOLO provides access to a selection of backbones based on the project's needs. This project experimented with the small (yolov5s) and extra-large (yolov5x) backbones and compared the results. When compared to Detectron, YOLO had several benefits that made it more beneficial for use in this project. First, YOLO separates the detected classes using color, making it more intuitive to see the oysters' activity at a glance. Secondly, the YOLO seemed to provide much better results for the same amount of training time. Finally, in the tests done, YOLOv5 was better at detecting small objects when compared to Detectron.

After experimenting with YOLOv5 and achieving decent results, this project shifted to additionally detect three different states of activity and infer each oyster's orientation. An oyster's orientation significantly influences the ability to detect activity, meaning that if orientation can be correctly detected, future models can be made more accurate. To attempt to solve this issue, YOLOv5_OBB was chosen for this project. Already having some experience with YOLOv5, YOLOv5_OBB seemed the natural choice for this problem. However, using this new model, the dataset format needed to be changed to use oriented bounding boxes. Therefore, the dataset was grown until it reached 1000 images, providing a good variety of oyster data. Unfortunately, the Roboflow export for the oriented bounding box format had issues and was incompatible with the model. To fix this, we developed a script to format the dataset properly and uploaded it to the Github repository for those interested in creating their versions of the dataset [40].

VI. Performance Analysis

After the object detector was trained, it was evaluated using some standard object detection metrics, recall, precision, mAP@0.5, and mAP@0.5:0.95. A model's loss can also measure how well a model performs on a dataset.

A. Precision & Recall

Before understanding precision or recall, it is crucial to understand the intersection over union (IoU) metric. IoU measures how close a predicted bounding box is to the ground truth bounding box. To calculate IoU, the intersection of the ground truth and predicted bounding box is divided by the union of the two boxes [41]. The model's performance can be reasonably measured by calculating the intersection over the union value for each bounding box.

FIGURE VIII

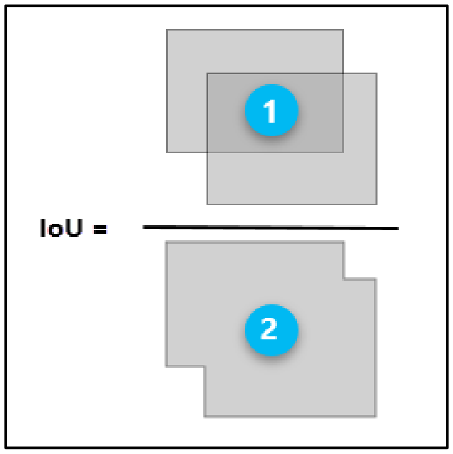

Fig 8. Above is a depiction of the intersection over union calculation. IoU is described as 1 (Intersection of the predicted and ground truth boxes) over 2 (the union of the same two boxes). Image from [41].

Usually, a good IoU value is said to be anything over 0.5; anything lower is considered poor. Using IoU, both recall and precision can be calculated. Using such an IoU threshold, the truth value can be determined for each prediction. If the model predicts a box with an IoU value over 0.5, it is called a true positive. On the other hand, if the value is less than 0.5, the prediction is considered a false positive. Lastly, if there is a ground truth box that the model fails to predict, it is labeled as a false negative [42]. The other possibility, true negative, is not needed in these calculations.

FIGURE IX

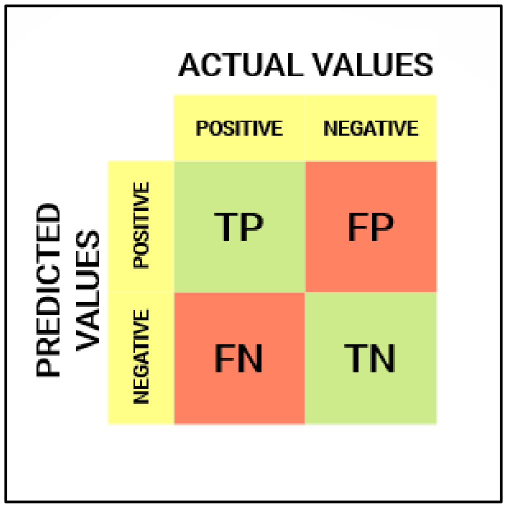

Fig 9. This figure shows all the possible values for a model's prediction from left to right, top to bottom, true positive, false positive, false negative, and true negative. Image obtained from [43].

By computing these values for each bounding box, recall and precision can be calculated for every image. The recall is computed by taking the ratio of the number of correctly predicted objects over the number of total objects.

$$R = \frac{(True\ Positives)}{(True\ Positives\ +\ False\ Negatives)} \tag{2}$$

Precision, however, measures how many of the model's predictions were correct. The model's precision can be calculated by taking the ratio of correct guesses over the total number of guesses [44].

$$P = \frac{(True\ Positives)}{(True\ Positives + False\ Positives)} \tag{3}$$

C. Mean Average Precision

By using precision and recall, a more accurate representation of the model's performance can be calculated, mean average precision (mAP). Before the mAP can be determined, average precision must be computed for each class. Then, the precision and recall values are graphed; taking the integral of this graph will return the AP value for the given class. By determining AP at a given IoU value for each class, the mAP can be computed by simply taking the average of all the AP values. mAP@0.5:0.95 is evaluated by calculating the mAP at IoU values ranging from 0.5 to 0.95 at intervals of 0.05 [45].

FIGURE X

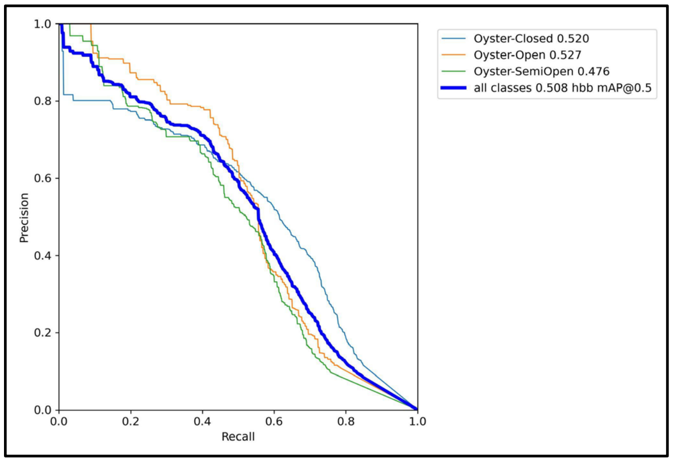

Fig 10. Above is the precision-recall curve for a 325-epoch training attempt on yolov5x.

Different models can be compared using a combination of these metrics, and the best performers can be easily identified.

D. Loss

Machine learning models learn by using a loss function. A loss function helps an algorithm grasp a specific dataset. A loss function can gradually reduce the model's error by measuring the distance between the ground truth value and the predicted value [46]. YOLOv5_OBB uses four different loss functions; the first, class loss, measures how well a model classifies the contents of a bounding box. Box loss is a metric to determine how closely the predicted bounding box fits the object. Objectness loss helps the model choose the predicted box with the best IoU value [47]. Finally, theta loss is used to accurately teach the model to predict an object's angle.

E. Model Comparison

Both yolov5n and yolov5x were trained, and their results were compared. When trained on the same number of epochs, the larger of the two models, yolov5x, took much longer to train but produced significantly better results.

TABLE I

	Precision	Recall	mAP 0.5	mAP 0.05:0.95
ylv5n	0.48	0.47	0.45	0.18
ylv5x	**0.53**	**0.62**	**0.53**	**0.28**

The two models trained for 325 epochs on the same dataset. The time needed to train yolov5n was much shorter than the time required for yolov5x. Higher numbers indicate better performance.

However, having a much more complex model structure, yolov5x began to overfit the data, increasing the model's loss. When training a model, the goal is to minimize the loss values, meaning that the model is correctly generalizing the dataset it is being fed. One possible method of reducing loss during training is introducing dataset augmentations. However, complex augmentations can adversely impact the model's performance, thus meaning it must be trained for more epochs.

TABLE II

	obj_loss	box_loss	cls_loss	theta_loss
ylv5n	**0.11**	0.048	**0.023**	**0.15**
ylv5x	0.15	**0.047**	0.024	0.16

Both trained on 325 epochs, and the metrics' loss values are displayed above. Having a less complex structure, yolov5n performed noticeably better in both obj_loss and cls_loss. Lower values are better. Values are rounded.

The developed program provides its users with a choice by training both models. The larger model makes much more accurate bounding boxes and predicts more off them; however, its inference time is much slower, running at about three frames per second on an Intel® Xeon® E5-2650 v4 24 Core CPU @ 2.20GHz with 32GB of RAM. On the other hand, while the smaller model may not be as accurate, it runs much faster, achieving above 30 frames per second, allowing it to be used for real-time inference. In addition to these results, both models were trained for longer durations to achieve better performance.

VII. Results

After the models were trained, we ran the models on various images of oysters to visualize the results. First, each oyster detected is labeled with its class which is also shown by the color. Following the label is the model's confidence score, a measure of how confident the model is that that bounding box contains the specified class. Next is the rotation score; the lower this number, the more likely it is that the oyster's opening is facing the camera, thus making it easier to classify its activity. Finally, the rotation angle specifies how many degrees the bounding box is rotated off-axis.

FIGURE XI

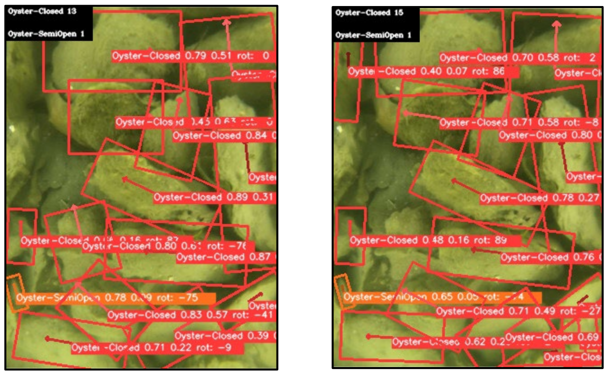

Fig 11. Right yolov5n left yolov5x. The number of each oyster detected can be seen in the top right. The arrows point parallel to the axis of orientation.

Yolov5x can detect more of the smaller oysters within the scene and have better fitting bounding boxes. In addition, it has better classification accuracy, having a better distinction between semi-open and open oysters.

A different example, inference on data that the model has not seen before can give a good insight into how a model generalizes what it has learned.

FIGURE XII

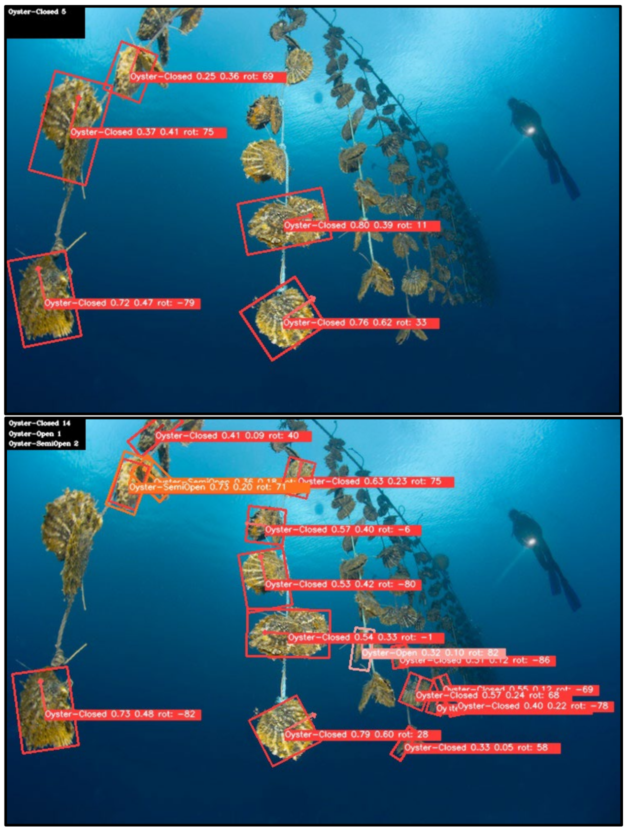
Oyster-Closed 5
Oyster-Closed 0.25 0.36 rot: 69
Oyster-Closed 0.37 0.41 rot: 75
Oyster-Closed 0.80 0.39 rot: 11
Oyster-Closed 0.72 0.47 rot: -79
Oyster-Closed 0.76 0.62 rot: 33
Oyster-Closed 14
Oyster-Open 1
Oyster-SemiOpen 2
Oyster-Closed 0.41 0.09 rot: 40
Oyster-SemiOpen 0.73 0.20 rot: 71
Oyster-Closed 0.63 0.23 rot: 75
Oyster-Closed 0.57 0.40 rot: -6
Oyster-Closed 0.53 0.42 rot: -80
Oyster-Closed 0.54 0.33 rot: -1
Oyster-Open 0.32 0.10 rot: 82
Oyster-Closed 0.57 0.24 rot: 68
Oyster-Closed 0.40 0.22 rot: -78
Oyster-Closed 0.73 0.48 rot: -82
Oyster-Closed 0.79 0.60 rot: 28
Oyster-Closed 0.33 0.05 rot: 58

Fig 12. Above yolov5n, below yolov5x. Image obtained from [48].

This image of an oyster farm differs from all the data the model was trained on. The looking-up view is something that the model has not seen before, and it does a respectable job identifying the oysters. Yolov5x identifies many more oysters and has much better fitting bounding boxes; however, yolov5n can detect one of the oysters that the larger model missed. Overall, the object detector shows promising results on new data, showing that it has potential for real-world applications.

However, the object detector does have some limitations. Firstly, the detector has trouble distinguishing empty oyster shells from live oysters. A possible solution would be to train the detector with an additional class, oyster shell, allowing the detector to distinguish between live and dead specimens. Secondly, the model may not find all the oysters within a scene; the number of detections made may be adjusted by reducing the confidence threshold, however, this has the adverse effect of increasing the number of false positive indications made. Additionally, the labels become unreadable in crowded scenes containing many oysters due to the vast amount of them on screen at once. Finally, the orientation detection is subject to the camera's perspective. The orientation of oysters pointed directly at the camera cannot be calculated accurately due to the lack of 3D information.

VIII. Future Work

Although this project made progress towards the final goal, a considerable amount of work still needs to be done before a complete project can be delivered. This section will briefly discuss what still needs to be done and possible routes for completing the project. While this research provided some orientation information, no calculations were done with the information. One possible next step for this project would be to take the information provided by this network and feed it into another model that uses orientation output to make more accurate classifications. Another path is to improve the orientation detection done by this model. Currently, the orientation is subject to the camera angle, and subjects pointed directly at the camera are not correctly classified. One solution is to use depth information such as that seen below.

FIGURE XIII

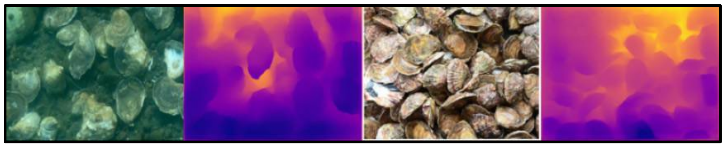

Fig 13. Figure a) An image of oysters underwater. Figure b) Depth inference run on figure a. Figure c) An image of oyster above water. Figure d) Depth inference run on figure c.

Initial results were obtained by passing images through the pre-trained DenseDepth, a deep learning model for extracting depth information from an image [49]. Deeper parts of the image are shown in orange, and the regions closer to the camera are shown in blue. Even without training this model on oyster images, these results were achieved; with more exploration into this topic, better results could likely be achieved. Secondly, 3-dimensional bounding boxes could improve orientation detection by using all three dimensions that an oyster can face. Using 3D bounding boxes could solve the issues previously described [50].

IX. Conclusion

As the human population grows, a large influx in global food demand will follow. Shellfish, specifically oysters, have been identified as a potential future food source to meet this demand. However, the oyster farming industry lacks the technological advancements needed to scale with newfound demand. This research sought to create an automated oyster detection system that will assist farmers in monitoring their oyster crops as a solution to this problem. By taking advantage of preexisting object detection models, this project used a custom compiled dataset to train an object detector to detect oysters in three different states of activity. In addition, this project attempted to improve previous work by using the oysters' orientation to assist in activity recognition. By taking advantage of these emerging technologies, farmers can have the upper hand when tackling one of the world's complex challenges.

X. References

[1] "Population," *United Nations*. [Online]. Available: https://www.un.org/en/global-issues/population. [Accessed: 21-Jul-2022].

[2] "Growing at a slower pace, world population is expected to reach 9.7 billion in 2050 and could peak at nearly 11 billion around 2100 | UN Desa Department of Economic and Social Affairs," *United Nations*. [Online]. Available: https://www.un.org/development/desa/en/news/population/world-population-prospects-2019.html. [Accessed: 22-Jun-2022].

[3] "Larger population, larger people: Humanity will require 80% more food by 2100," *Population Matters*, 22-Feb-2021. [Online]. Available: https://populationmatters.org/news/2019/12/larger-population-larger-people-humanity-will-require-80-more-food-2100#:~:text=2100%20%7C%20Population%20Matters-,Larger%20population%2C%20larger%20people%3A%20humanity%20will%20require,80%25%20more%20food%20by%202100&text=A%20new%20study%20shows%20that,global%20food%20demand%20to%20soar. [Accessed: 21-Jul-2022].

[4] *Enyue Annie Lu*. [Online]. Available: http://faculty.salisbury.edu/~ealu/REU/Projects.html. [Accessed: 22-Jun-2022].

[5] W. W. Billiot, "The Ancient Oyster," *Country Roads Magazine*, 23-Jan-2015. [Online]. Available: https://countryroadsmagazine.com/outdoors/knowing-nature/the-ancient-oyster/. [Accessed: 22-Jul-2022].

[6] "Impact of technology on agriculture," *National Geographic Society*. [Online]. Available: https://education.nationalgeographic.org/resource/impact-technology-agriculture. [Accessed: 22-Jul-2022].

[7] M. N. Azra, V. T. Okomoda, M. Tabatabaei, M. Hassan, and M. Ikhwanuddin, "The contributions of shellfish aquaculture to global food security: Assessing its characteristics from a future food perspective," *Frontiers*, 01-Jan-1AD. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fmars.2021.654897/full. [Accessed: 22-Jul-2022].

[8] B. Sadrfaridpour, Y. Aloimonos, M. Yu, Y. Tao, and D. Webster, "Detecting and counting oysters," *arXiv.org*, 20-May-2021. [Online]. Available: https://arxiv.org/abs/2105.09758. [Accessed: 22-Jul-2022].

[9] "Oyster restoration," *Chesapeake Bay Foundation*. [Online]. Available: https://www.cbf.org/about-cbf/our-mission/restore/oyster-restoration/. [Accessed: 24-Jun-2022].

[10] University of Maryland Eastern Shore, "Enee 422 introduction to machine learning," *University of Maryland Eastern Shore*. [Online]. Available: https://stg15.umes.edu/Engineering/DynPage/ENEE-422-Introduction-to-Machine-Learning/. [Accessed: 01-Aug-2022].

[11] Jwtownsend, "ENCE 452 Artificial Intelligence: University of Maryland Eastern Shore," *Engineering Program*, 25-Feb-2022. [Online]. Available: https://wwwcp.umes.edu/engineering/ence-452-artificial-intelligence/. [Accessed: 22-Jul-2022].

[12] "Non Technical Summary," *Transforming shellfish farming with smart technology and management practices for sustainable production - UNIV of Maryland*. [Online]. Available: https://portal.nifa.usda.gov/web/crisprojectpages/1023149-transforming-shellfish-farming-with-smart-technology-and-management-practices-for-sustainable-production.html. [Accessed: 22-Jul-2022].

[13] "Course information," *UMass Lowell*. [Online]. Available: https://www.uml.edu/catalog/courses/engn/4010. [Accessed: 01-Aug-2022].

[14]

[15] V. J. Major, N. Jethani, and Y. Aphinyanaphongs, "Estimating real-world performance of a predictive model: A case-study in predicting mortality," *JAMIA open*, 26-Apr-2020. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7382635/. [Accessed: 01-Aug-2022].

[16] B. Sadrfaridpour, Y. Aloimonos, M. Yu, Y. Tao, and D. Webster, "Detecting and counting oysters," *arXiv.org*, 20-May-2021. [Online]. Available: https://arxiv.org/abs/2105.09758. [Accessed: 22-Jul-2022].

[17] "Using underwater robots to detect and count oysters," *Using underwater robots to detect and count oysters | Institute for Systems Research*. [Online]. Available: https://isr.umd.edu/news/story/using-underwater-robots-to-detect-and-count-oysters. [Accessed: 01-Aug-2022].

[18] J. Chen, "Neural network definition," *Investopedia*, 17-Jun-2022. [Online]. Available: https://www.investopedia.com/terms/n/neuralnetwork.asp#:~:text=A%20neural%20network%20is%20a,organic%20or%20artificial%20in%20nature. [Accessed: 26-Jul-2022].

[19] N. Lang, "Using convolutional neural network for Image Classification | Detail: Convolutional Layer," *Medium*, 28-Apr-2022. [Online]. Available: https://towardsdatascience.com/using-convolutional-neural-network-for-image-classification-5997bfd0ede4. [Accessed: 26-Jul-2022].

[20] N. Lang, "Using convolutional neural network for Image Classification | Detail: Pooling Layer," *Medium*, 28-Apr-2022. [Online]. Available: https://towardsdatascience.com/using-convolutional-neural-network-for-image-classification-5997bfd0ede4. [Accessed: 26-Jul-2022].

[21] N. Lang, "Using convolutional neural network for Image Classification | Detail: Fully-Connected Layer," *Medium*, 28-Apr-2022. [Online]. Available: https://towardsdatascience.com/using-convolutional-neural-network-for-image-classification-5997bfd0ede4. [Accessed: 26-Jul-2022].

[22] "Object localization," *Frontiers*. [Online]. Available: https://www.frontiersin.org/research-topics/28766/object-localization. [Accessed: 26-Jul-2022].

[23] "Introduction," *YOLOv5 Documentation*. [Online]. Available: https://docs.ultralytics.com/. [Accessed: 22-Jul-2022].

[24] T. A. I. Team, "Yolo v5 -explained and demystified," *Towards AI*, 01-Jul-2020. [Online]. Available: https://towardsai.net/p/computer-vision/yolo-v5%E2%80%8A-%E2%80%8Aexplained-and-demystified. [Accessed: 22-Jul-2022].

[25] Jeremy Jordan, "An overview of object detection: One-stage methods," *Jeremy Jordan*, 18-Sep-2018. [Online]. Available: https://www.jeremyjordan.me/object-detection-one-stage/#yolo. [Accessed: 22-Jul-2022].

[26] "Papers with code - panet explained," *Explained | Papers With Code*. [Online]. Available: https://paperswithcode.com/method/panet. [Accessed: 22-Jul-2022].

[27] Ultralytics, "Overview of model structure about Yolov5 · issue #280 · ultralytics/yolov5," *GitHub*. [Online]. Available: https://github.com/ultralytics/yolov5/issues/280. [Accessed: 22-Jul-2022].

[28] X. Yu, M. Lin, J. Lu, and L. Ou, "Oriented object detection in aerial images based on area ratio of parallelogram," *arXiv.org*, 08-Nov-2021. [Online]. Available: https://arxiv.org/abs/2109.10187. [Accessed: 25-Jul-2022].

[29] X. Yu, M. Lin, J. Lu, and L. Ou, "Oriented object detection in aerial images based on area ratio of parallelogram," *arXiv.org*, 08-Nov-2021. [Online]. Available: https://arxiv.org/abs/2109.10187. [Accessed: 01-Aug-2022].

[30] "Underwater Image Recognition Detector using Deep Convnet," *IEEE Xplore*. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9056058. [Accessed: 24-Jun-2022].

[31] *Inquiry about oyster images*, 06-Jun-2022.

[32] *Roboflow*. [Online]. Available: https://app.roboflow.com/newoysters/threestateoyster/19. [Accessed: 14-Jul-2022].

[33] "Non Technical Summary," *Transforming shellfish farming with smart technology and management practices for sustainable production - UNIV of Maryland*. [Online]. Available: https://portal.nifa.usda.gov/web/crisprojectpages/1023149-transforming-shellfish-farming-with-smart-technology-and-management-practices-for-sustainable-production.html. [Accessed: 22-Jul-2022].

[34] "The Essential Guide to data augmentation in Deep Learning," *V7*. [Online]. Available: https://www.v7labs.com/blog/data-augmentation-guide. [Accessed: 25-Jul-2022].

[35] "Non Technical Summary," *Transforming shellfish farming with smart technology and management practices for sustainable production - UNIV of Maryland*. [Online]. Available: https://portal.nifa.usda.gov/web/crisprojectpages/1023149-transforming-shellfish-farming-with-smart-technology-and-management-practices-for-sustainable-production.html. [Accessed: 22-Jul-2022].

[36] Real Python, "Python vs C++: Selecting the right tool for the job," Real Python, 19-Jun-2021. [Online]. Available: https://realpython.com/python-vs-cpp/#summary-python-vs-c. [Accessed: 25-Jul-2022].

[37] "A complete list of the best laptop for Machine Learning in 2019," *Edureka*, 05-Jan-2022. [Online]. Available: https://www.edureka.co/blog/best-laptop-for-machine-learning/#:~:text=RAM%3A%20A%20minimum%20of%2016,powerful%20and%20delivers%20High%20Performance. [Accessed: 25-Jul-2022].

[38] *Google colab*. [Online]. Available: https://colab.research.google.com/?utm_source=scs-index. [Accessed: 14-Jul-2022].

[39] Facebookresearch, "Facebookresearch/Detectron2: Detectron2 is a platform for object detection, segmentation and other visual recognition tasks.," *GitHub*. [Online]. Available: https://github.com/facebookresearch/detectron2. [Accessed: 24-Jun-2022].

[40] Facebookresearch, "Detectron2/MODEL_ZOO.MD at main · facebookresearch/detectron2," *GitHub*, 21-Jul-2021. [Online]. Available: https://github.com/facebookresearch/detectron2/blob/main/MODEL_ZOO.md. [Accessed: 24-Jun-2022].

[41] Zenny00, "Zenny00/reu-oyster_orientation: This repository contains the code for the detecting the multiple states of oyster activity and orientation using deep learning image processing and computer vision algorithms project as part of the NSF REU 2022 program," *GitHub*. [Online]. Available: https://github.com/Zenny00/REU-Oyster_Orientation. [Accessed: 01-Aug-2022].

[42] "How the compute accuracy for Object Detection Tool Works," How the Compute Accuracy For Object Detection tool *Interpret model results* works-ArcGIS Pro | Documentation. [Online]. Available: https://pro.arcgis.com/en/pro-app/2.8/tool-reference/image-analyst/how-compute-accuracy-for-object-detection-works.htm#:~:text=Recall%E2%80%94Recall%20is%20the%20ratio,the%20recall%20is%2075%20percent.&amp;text=F1%20score%E2%80%94The%20F1%20score,of%20the%20precision%20and%20recall. [Accessed: 27-Jul-2022].

[43] "How the compute accuracy for Object Detection Tool Works," How the Compute Accuracy For Object Detection tool *Accuracy Outputs* works-ArcGIS Pro | Documentation. [Online]. Available: https://pro.arcgis.com/en/pro-app/2.8/tool-reference/image-analyst/how-compute-accuracy-for-object-detection-works.htm#:~:text=Recall%E2%80%94Recall%20is%20the%20ratio,the%20recall%20is%2075%20percent.&amp;text=F1%20score%E2%80%94The%20F1%20score,of%20the%20precision%20and%20recall. [Accessed: 27-Jul-2022].

[44] "Confusion matrix for Machine Learning," *Analytics Vidhya*, 14-Jun-2022. [Online]. Available: https://www.analyticsvidhya.com/blog/2020/04/confusion-matrix-machine-learning/. [Accessed: 27-Jul-2022].

[45] R. Khandelwal, "Evaluating performance of an object detection model," *Medium*, 06-Jan-2020. [Online]. Available: https://towardsdatascience.com/evaluating-performance-of-an-object-detection-model-137a349c517b. [Accessed: 27-Jul-2022].

[46] "Mean average precision (MAP) explained: Everything you need to know," *V7*. [Online]. Available: https://www.v7labs.com/blog/mean-average-precision#:~:text=Mean%20Average%20Precision(mAP)%20is%20a%20metric%20used%20to%20evaluate,values%20from%200%20to%201. [Accessed: 27-Jul-2022].

[47] R. Parmar, "Common loss functions in machine learning," *Medium*, 02-Sep-2018. [Online]. Available: https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23. [Accessed: 28-Jul-2022].

[48] U. Almog, "Yolo V3 explained," *Medium*, 13-Oct-2020. [Online]. Available: https://towardsdatascience.com/yolo-v3-explained-ff5b850390f. [Accessed: 28-Jul-2022].

[49] "Fiji," *Sustainable Pearls*. [Online]. Available: http://www.sustainablepearls.org/pearls/pearl-farming-around-the-world/fiji/. [Accessed: 28-Jul-2022].

[50] Ialhashim, "Ialhashim/DenseDepth: High quality monocular depth estimation via transfer learning," *GitHub*. [Online]. Available: https://github.com/ialhashim/DenseDepth. [Accessed: 28-Jul-2022].

[51] Skhadem, "SKHADEM/3D-BoundingBox: Pytorch implementation for 3D bounding box estimation using deep learning and Geometry," GitHub. [Online]. Available: https://github.com/skhadem/3D-BoundingBox. [Accessed: 28-Jul-2022].